Assessing Information Trustability in a Secure Web Services Environment

THESIS

Charles G. Penner, Captain, USAF

AFIT/GCS/ENG/05-14

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or U.S. Government.

AFIT/GCS/ENG/05-14

Assessing Information Trustability in a Secure Web Services Environment

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Charles G. Penner, B.S.

Captain, USAF

March 2005

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT/GCS/ENG/05-14

Assessing Information Trustability in a Secure Web Services Environment

Charles G. Penner, B.S.
Captain, USAF

Approved:

/signed/	21 Mar 2005
Lt Col Michael Talbert, PhD (Chairman)	date
/signed/	21 Mar 2005
Maj Robert Graham (Member)	date
/signed/	21 Mar 2005
Maj Christopher Mayer (Member)	date

AFIT/GCS/ENG/05-14

Abstract

Decisions are made based on available information. A decision support system endeavors to provide information that is timely, accurate, and trustable. Information gathered from secure web service transactions has attributes that can be used to assess a level of trustability. The trust assessments enable a decision maker to determine a basis for confidence in the information presented from the web service. Existing trust assessment models do not provide a way to determine from a particular trust assessment what information attributes contributed to its computation. The present work creates trust values that retain and denote meaning, allowing a decision maker to see specifically what factors influenced the information trust assessment. Also central to this work is interpretation of the trust assessments. The interpretation model allows users to specify the amount of allowable tolerance for reduced trustability in the decision being made. This "dial-a-trust" allows the interpretation to be scaled relative to the impact of the decision. The trust assessment values, along with their interpretations, allow both human and machine-based decision makers to determine whether information is trustable enough for the needs of the decision being made.

Acknowledgements

Above all others I would like to thank my Savior Jesus Christ. None of this work would have been achievable without His strength. Next I would like to thank my wife for enduring the many long hours that prevented me from spending as much time with her and our daughter as we all would have liked. I would also like to thank my advisor Lt Col Talbert, his advice and guidance was vital both to the completion of this thesis, and AFIT as a whole. Thanks are also due to my committee members, Maj Graham and Maj Mayer; their comments and suggestions truly helped make for a better product. Finally, a special thanks to K0, K2, and Pat; their friendship was invaluable throughout the AFIT experience. Long live the lunch club!

Charles G. Penner

Table of Contents

Page

List of Figures

List of Tables

Table Page

List of Abbreviations

Assessing Information Trustability in a Secure Web Services Environment

I. Introduction

The need for timely, accurate, and trusted information is a pervasive force that drives every decision we make. Accordingly we must be able to ascertain the level with which a given set of information can be trusted. If we consider the notion that trustability has tangible attributes, then we suggest that a relative trustability value can be assessed for an information set $\mathfrak{I}$. If we restrict our focus to the communications domain of secure web services, we submit that a relevant set of indicators can be identified and used to approximate such a value. This research examines how trustability can be assessed for $\mathfrak{I}$ in a secure web services environment.

1.1 Background

The concept of ensuring high trustability of information is not a new one. It seems intuitive that any organization tasked with critical decision making would be interested exclusively in information from trusted sources. Distributed computing and web services have enabled information to be consolidated from multiple sources, allowing differences in protocols and standards to be overcome. While this relative independence from specific protocols is a welcome change for system engineers, a new problem is introduced as information retrieval experts work to ensure that the picture presented from multiple sources is indeed a trustable one. As noted by others [14], there are several definitions of trust that have been put forth in literature (see Section 2.2 for a brief survey of several).

One of the leading definitions of trust has been labeled *Information Integrity* (I*I) by Martin Bariff and Salimol Thomas of the Information Integrity Coalition (IIC). I*I is defined as "the correctness of information, includ[ing] the accuracy, consistency, and reliability of information domains (content, process, & system) of an enterprise" [13]. Certainly if we are determining

the trustability of some information, we must be able to ascertain the measure of its accuracy, consistency, and reliability. Yet to say that these dimensions paint a complete picture of trustability is premature. This work proposes additional indicators that give greater insight into how trustable a particular set of information is.

Given a focus on secure web services, a security framework must be given with which to provide an environment for trustability evaluation. Until recently, no such framework existed among the various standards bodies contributing to the field of web services. However, in April 2004 the Organization for the Advancement of Structured Information Standards (OASIS) ratified the Web Services Security (WS-Security) standard. WS-Security proposes "a standard set of SOAP extensions that can be used when building secure Web services to implement message content integrity and confidentiality" [9]. This progress was recognized by Gartner Inc. within days of its ratification, recommending that enterprises should adopt WS-Security for all "across-the-wall Web service deployments" [26].

1.2 Methodology

Along with defining I*I, Bariff and Thomas propose a framework within which to calculate the integrity of a given set of information. Using their domains (content, process, and system) and attributes (accuracy, consistency, and reliability), I*I is calculated by a series of multiplication functions for each, resulting in a final value between 0 and 1. While providing an easy way to calculate trustability, the framework does not provide a way to distinguish between two like values. As demonstrated in Section 2.3.1, the same trust value can be generated from a wide range of input. One cannot see the contributing factors in the final value, which we believe decreases its usefulness to the decision making process.

Individual contributing factors should play a role in the decisions made. In order to make the best decision, insight must be given into the composition of the trust value. If the accuracy

attribute is low, then the information probably should not be trusted, and decisions must be made accordingly. However, if the consistency attribute is suddenly lowered, then a decision needs to be made whether or not to trust the information; perhaps there are events transpiring that have resulted in a valid change within the information space. In such a case, we can see that the low consistency rating is very useful to decision makers because it allows them to focus on areas that are affecting the trustability.

The notion of domains and attributes is a useful concept, and can be used to gather indicators used to calculate trust for $\mathfrak{I}$. This work proposes a method by which a computed trustability value carries with it meaningful semantics. Whereas the I*I framework multiplies the indicators to compute a single composite value eliminating the individual factors, the present work retains each individual trust indicator that contributes to the overall trust. All trustability indicators are encoded into a representation in such a way that any value can be decomposed into a meaningful, repeatable interpretation of how it was derived. In this way we can easily determine what set of inputs yielded the received trust assessment.

This representation is implemented as an extension to WS-Security web services. As SOAP services are accessed by a client, trustability indicators are evaluated, and the trust is computed and stored in a database accessible only to the web service. When data is retrieved by information consumers, the calculated trust values are retrieved, displayed, and interpreted for the consumer to determine the basis for confidence in the presented data. Also available is a means with which to express a level of acceptable tolerance of the computed trust.

1.3 Overview

Having laid a foundation, the remainder of this document presents supporting evidence. Chapter II outlines the current state of secure web services. Several definitions of trust are presented and the I*I framework is given in more detail. Chapter III is an expanded methodology, where in-

depth coverage is given to calculating, encoding, and interpreting trust values. Chapter IV details results from a test implementation using a SOAP service secured with WS-Security. Chapter V presents conclusions and conjectures on trustability as well as possibilities for future work.

II. *Literature Review*

This chapter lays some ground work for pursuing the goals stated in Chapter I. First, the current state of secure web services is examined. Second, several definitions of trust are presented and discussed. Finally, the I*I framework is given in greater detail, along with some example computations.

2.1 *The State of Web Services*

The term "web services" is broad in scope, but is generally accepted to be a set of open standards that enable software applications to be accessed over Internet protocols. There are three standards that are commonly used: SOAP (formerly known as Simple Object Access Protocol), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). SOAP is the protocol that enables applications to communicate across different platforms. WSDL is used to describe a particular service and its capabilities, and UDDI allows services to be advertised and found on the Internet or local networks. The present work focuses exclusively on SOAP; WSDL and UDDI do not play a role in the trustability assessment methodology presented in Chapter III.

Since their introduction in 2000, SOAP-based web services have received much attention across the industry. The lightweight nature of the protocol and openness of the standards used have made them very appealing to organizations. A quick search of several leading news search engines for "SOAP web services" in February, 2005 returned several hundred news articles from each site [10] [11] - clearly web services is a topic that the industry is interested in.

In the February 2004 issue of *Communications of the ACM*, Conan C. Albrecht states that "although it was initially designed by Microsoft, UserLand, and Developmentor as a protocol for the BizTalk architecture, SOAP quickly gained momentum as IBM, Sun, Lotus, CommerceOne, and other proponents took interest in its development" [12]. The involvement of these industry leaders

didn't stop with the initial SOAP specification; IBM, Microsoft, and Sun are all listed as contributors to the WS-Security specification [9]. The fact that there are many other organizations involved with the SOAP and WS-Security specifications and that the specifications have been contributed to the World Wide Web Consortium (W3C) are indicators that SOAP is a true emerging standard, not just the work of a few isolated companies. In 2003 IEEE sponsored the first International Conference on Web Services, another sign from the industry that web services have grown and continue to grow in popularity.

As previously mentioned, when SOAP was initially proposed it stood for Simple Object Access Protocol (the acronym has since been removed from the name). When compared to its predecessors, it is indeed much simpler in both implementation and use. Unlike other distributed system protocols, SOAP is not bound to any particular architecture or technology set. Jim Clune and Dr. Adam Kolawa of Parasoft Corporation point to the Object Management Group's Internet Inter-Object Request Broker Protocol (IIOP) being the "underlying transport mechanism used by the Common Object Resource Broker Architecture (CORBA)" [16]. While IIOP and CORBA may provide valuable interoperability capabilities, they are tightly coupled, each requiring the other for implementation and deployment. Freedom from protocol lock-in makes the use of SOAP web services appealing to organizations that are considering deploying them.

Amit Sheth and John A. Miller from the University of Georgia agree with this idea [25]. From its beginnings CORBA was very complex, requiring experienced developers in order for it to be useful. Most businesses who used CORBA relied on expensive object request brokers, making it difficult to start new projects with it. In stark contrast is the web services model, where the key standards are free. Due to the available choices, in most cases a web services project can be "developed with essentially no initial technology cost" [25]. While they provide a simple framework and low cost of entry, web services are also extensible, able to increase in complexity as

needed to support enhanced functionality. This overall ease of use with a wide range of capabilities are what has helped SOAP based web services gain industry acceptance as quickly as they have.

2.1.1 The Security Void in SOAP. Despite all the attention being received by web services, an important aspect that is not addressed within the SOAP specification is security. In the interest of simplicity and extensibility, features common to most distributed systems were omitted from the specification. The W3C SOAP specification states that it "does not directly provide any mechanisms for dealing with access control, confidentiality, integrity, and non-repudiation. Such mechanisms can be provided as SOAP extensions using the SOAP extensibility model..." [8]. While this focus on simplicity lends itself to easy development, it also increases the ease of potential compromises.

These potential compromises are highlighted by Clune and Kolawa in [16]. The access ports that are so easily available for web services use are also potential access points for hackers and viruses. "Depending on how your software is configured, a remote operator could access your system and provide instructions to your server" [16]. Obviously this is a situation which should be avoided at all costs. While it is beneficial to the development cycle to have an open and flexible system, if not properly secured the ramifications could be detrimental to a web services-based system.

2.1.2 The Need for SOAP Security. According to David Geer in the October 2003 issue of *Computer*, mechanisms that have typically been used for securing web-based communications do not scale sufficiently for SOAP web services. Secure Sockets Layer (SSL) has been the primary means, using public and private key encryption combined with digital certificates to authenticate users. Use of SSL has been supplanted by the newer Transport Layer Security (TLS). Geer states that "TLS provides encryption-based connection security, and lets servers and clients authenticate one another and determine the cryptographic algorithms and keys that can be used for data transfer" [17]. While they have provided sufficient security thus far, due to the fact that they must

decrypt data every time it arrives at an intermediary, and encrypt it before sending it on again, they do not "scale well to complex, high-volume transactions" [17].

Satoshi Hada and Hiroshi Maruyama of IBM Research agree that layer level security measures are not sufficient for securing SOAP web services. As an alternative they propose several arguments in favor of using message layer security [18]: end-to-end security, application independence, transport independence, and security of stored messages.

1. **End-to-end Security:** Because SOAP is not bound to any particular transport protocol, it may pass through a number of intermediaries between the originator and the destination (see Figure 2.1). An intermediary must process any SOAP message header entries intended for it, removing them before forwarding the message. New header entries for other intermediaries may also be inserted before forwarding. Since the intermediaries have access to the message for processing, the transmission security provided by SSL and TLS are not sufficient. An untrusted intermediary could tamper with a message before forwarding it on. A secure transmission line provides no benefit if untrusted third parties are maliciously handling messages along the route.

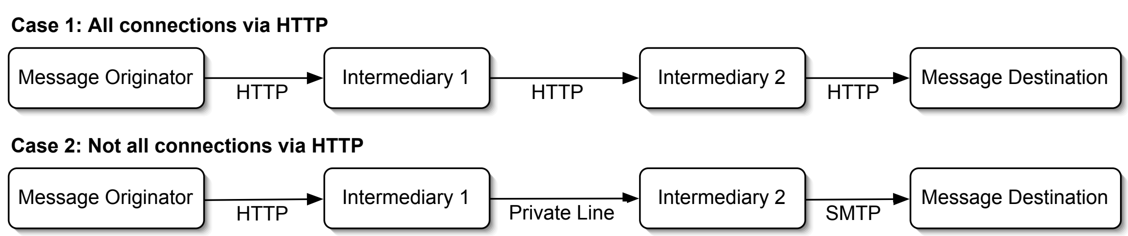

Figure 2.1: SOAP transmission with intermediaries [18]

2. **Application Independence:** Hada and Maruyama claim that in order to achieve true end-to-end security, it must be implemented at the application level. If there is any point where messages are transmitted in plain text, it is subject to attack. Integrating cryptographic functionality into applications is not something that is easily done without compromising application security. While there are a number of viable cryptographic libraries available,

their flexibility requires a high level of understanding in order to successfully ensure security. The authors argue for a "standardized, application-independent security layer" [18] to ensure adequate protection while removing the need for cryptographic expertise.

3. **Transport Independence:** Even if we consider all communication links secure, and trust all intermediaries in Case 2 of Figure 2.1, it is still not adequate to ensure SOAP security. As previously mentioned, and illustrated in Figure 2.1, intermediaries will often forward messages using different protocols from that which the message was received on. When changing protocols, any security information that exists (e.g., authenticity of the message originator) must be translated to a format that the next protocol can understand. This process can be tedious and complex, allowing a greater possibility for message tampering. If the security is handled at the message layer then the number of protocols encountered is irrelevant, and the need for TLS is isolated to what it is intended for - transmission security.

4. **Security of Stored Messages:** Were SOAP security to be implemented exclusively at the transport layer, there would be absolutely no security for messages that are stored after transactions have completed. In the same way that TLS is irrelevant when an intermediary dismantles messages for header information, when a message arrives at its final destination TLS no longer plays a role in the security of the message. In a situation where data is never stored at all this would not be an issue. However, in many situations messages are logged for analysis and auditing. This area of concern is of particular interest to this work, as it evaluates messages for their trustability. Message level security ensures the security of messages regardless of their current state (in transit or stored).

In the January 2004 issue of *BT Technology Journal*, Kearney *et al* [20] also favor message level security over transport layer security for securing SOAP messages. Consistent with other evaluations of TLS, they agree that it provides good security during message transmission. "However, it conveys nothing about any processing done at either end of the connection" [20]. Dispelling the

idea that intermediaries won't be used in practice, they point out that the "SOAP specifications clearly envisage the possibility of SOAP networks rather than just point-to-point connections" [20]. According to the specification, a node involved in a SOAP transaction may be one of three things: the sender, the receiver, or an intermediary. This allows the possibility for applications at either end of an SSL connection to act as intermediaries, forwarding messages to unknown, non-secure locations. Except in the simplest of situations, the authors recommend message level security to combat these issues.

One of the purported benefits of SOAP is its ability to penetrate through and be accessed from behind firewalls. Because it is text based and often accessed via HTTP, port 80 is commonly used. Since web servers utilize port 80 for their traffic, developers can count on it being open on any firewalls their SOAP traffic may encounter. While this ease of access is enticing, it is also a potential trouble area. Most web traffic results in a human-readable page that is fairly harmless, but as Kearney *et al* remind us, "a SOAP message is designed to trigger some activity in the system receiving it, and this is open to abuse as well as legitimate use" [20].

This is a recurring theme throughout the literature that is perhaps most vehemently argued by industry leading security expert Bruce Schneier. In the June 15, 2000 issue of his monthly Crypto-Gram Newsletter, he laments the arrival of SOAP. Because there is no security standard built into the SOAP spec, he states: "It's a pretty simple bet that different people will bungle any embedded security in different ways, leading to different holes on different implementations. SOAP is going to open up a whole new avenue for security vulnerabilities" [24]. While it's easy to see that firewall holes don't enhance security, Schneier seems a bit extreme with his final proclamation on the issue, saying "protocols that sneak ... through [firewalls] are not what's wanted" [24].

Albrecht takes a slightly different stance on this issue, merely raising the point that "as it becomes more ubiquitous, SOAP may increasingly find itself in the sights of security personnel" [12]. He encourages organizations to find their own balance between security and functionality.

The most strenuous security plan allows minimal traffic into the organization, limiting overall functionality. At the other end of the spectrum are firewalls that allow multiple port ranges and/or protocols into the internal network to give expected functionality to applications. In his eyes, SOAP is a possible solution to retaining security while allowing functionality. No additional ports are required to be opened if the organization is using port 80 for their web servers already, and SOAP applications are able to tunnel through firewalls as needed.

He cautions against blindly proceeding with this type of configuration though, warning that soon "network security administrators may see it as an 'end run' around security" [12]. Since SOAP traffic has its own unique content type, Albrecht advises that firewalls can be configured to monitor traffic for specific SOAP methods being invoked as well as other functionality. While not commonly used in current SOAP applications, filtering is bound to become more common as the number of SOAP applications increase. Albrecht believes firewall content filtering has the potential to "undermine one of SOAP's primary benefits" [12] of being able to pass through firewalls. He concludes with no solid fix to the problem, in essence concurring with Geer, Hada and Maruyama, and Kearney *et al* that something new is needed to realistically secure SOAP messages and allow expected functionality.

2.1.3 Securing Web Services. With all the warnings about the lack of security in SOAP web services, there is much work afoot within the industry to remedy the problem. According to Kearney *et al*, "there are several standards bodies active in this space, notably W3C and OASIS" [20]. They discuss five of the more significant initiatives: the WS-X series of specifications, W3C activities, OASIS, the Liberty Alliance, and the Web Services Interoperability (WS-I) organization.

1. **WS-X series of specifications:** This is a family of specifications put forth by IBM, Microsoft, and others in conjunction with the Global XML Web Services Architecture. There are seven specifications proposed: WS-Security, WS-Trust, WS-Security Policy, WS-SecureConversation, WS-Federation, WS-Privacy, and WS-Authorization. WS-Security is the foundational layer

for the other specifications. Despite what the name may imply, it is not an attempt to secure all aspects of web services. Rather, "it is a building block that is intended to be used in conjunction with other Web Services and application-specific protocols to accommodate a wide variety of security models" [20]. Figure 2.2 (from the IBM and Microsoft roadmap) gives a pictorial representation of how the different specifications fit together and their progress to-date. Although their scope is broad, Kearney *et al* are quick to point out that the WS-X series are "still relatively immature, and it remains to be seen to what extent they are taken up in practice" [20].

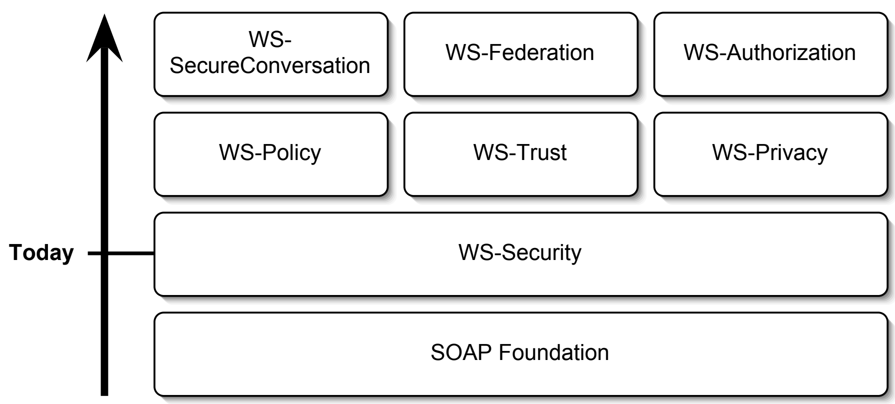

Figure 2.2: Web Services Security Specifications [19]

2. **W3C activities:** W3C has tended to serve as the focal point for fundamental web services standards (e.g. SOAP and WSDL). Several key standards currently supported are XML Encryption, XML Signature, and XML Key Management. Under the WS-Security specification, security metadata similar to what is found in XML Encryption and XML Signature are attached to SOAP header messages [17]. Another key contribution of W3C is the Semantic Web and Web Ontology Working Group. "Although defining and interpreting semantics has received little attention in current Web Services applications, it will become vital as Web Services networks become larger and more complex" [20].

3. **OASIS:** There are several technical committees (TCs) of importance that have come out of OASIS. Of particular interest to this work is the WS-Security TC which was established in order to further develop and promote the WS-Security specification as an industry wide standard.

4. **Liberty Alliance:** Formed in September 2001, the goal of the Liberty Alliance Project is to "develop open standards for federated network identity management and identity-based services" [20]. Its origins came as a response to Passport, the identity management technology from Microsoft. There are many attractive features contained within the project. Additionally there are areas of "overlap and 'competition' between the WS-X security specifications" [20] and the Liberty Project.

5. **WS-I organization:** Differing slightly from the other standards mentioned, the goal of WS-I is to promote web service interoperability by defining profiles to illustrate how to apply web service standards. WS-I Basic is the initial profile which covers XML Schema 1.0, SOAP 1.1, WSDL 1.1, and UDDI 2.0. It has recently formed a Basic Security group, examining the "use of HTTPS, SOAP attachment security, and the OASIS WSSTC specifications (i.e. WS-Security)" [20].

2.2 Definitions of Trust

Within the literature many definitions of trust are available, all with their own connotations of what it means to trust some set of information. Generally speaking, there are two broad categories of trust defined. There is the field of thought which aligns the definition with that from the field of data quality. In other words, trust is a function of the quality of the data. The alternative field of thought has its roots in cryptography, answering the question: "how sure can I be that this data has not been tampered with"? Also presented here is a concept termed *The Trust Management Philosophy*, an approach that blends quality-based trust with cryptographic-based trust.

2.2.1 Quality-Based Trust. Definitions of quality based trust vary depending on which body or organization is defining them. Despite the variances, there is a common ground among them that we will focus on. Madhavan K. Nayar from the IIC, Peter Chen of Louisiana State University, and Pipino *et al* provide several definitions that we examine here.

According to Nayar, while Information Integrity (I*I) is not explicitly data quality, it is "directly concerned with the *accuracy*, *consistency*, and *reliability* of information with its supporting processes and system" [22] (emphasis added). These attributes bring important contributions to the trust determination process. Each can be applied in the domains of *content*, *process*, and *system*. The I*I framework defines a methodology by which the attributes can be evaluated in an objective manner. Using this methodology, the attributes and domains determine the "trustworthiness or dependability of information" [22]. The I*I framework is given in more detail in Section 2.3.

In the April 2002 issue of *Communications of the ACM*, Pipino *et al* discuss a number of data quality dimensions set in two categories, "... subjective perceptions of the individuals involved with the data, and the objective measurements based on the data set in question" [23]. Perhaps the most relevant to our discussion of trust is the dimension of *believability*, "the extent to which data is regarded as true and credible" [23]. Incorporated in its measurement is individual assessments of source credibility, comparison of data to accepted standards, and previous experiences. The complete list of dimensions and their definitions is shown in Table 2.1.

Chen also has an important contribution to the definition of quality based trust. He terms calculating trustability as an *Information Validity Assessment*, composed of three factors: reliability of the database hardware and software, freshness of the data, and believability of the data [15]. Database reliability and data believability overlap in their definitions, both having to do with the source of the data and its reliability and believability. The impact of data freshness is important; data that has outlived its shelf life should be trusted less than data that is considered fresh and up to date.

Table 2.1: Data Quality Dimensions [23]

Dimensions	Definitions
Accessibility	the extent to which data is available, or easily and quickly retrievable
Appropriate Amount of Data	the extent to which the volume of data is appropriate for the task at hand
Believability	the extent to which data is regarded as true and credible
Completeness	the extent to which data is not missing and is of sufficient breadth and depth for the task at hand
Concise Representation	the extent to which data is compactly represented
Consistent Representation	the extent to which data is presented in the same format
Ease of Manipulation	the extent to which data is easy to manipulate and apply to different tasks
Free-of-Error	the extent to which data is correct and reliable
Interpretability	the extent to which data is in appropriate languages, symbols, and units, and the definitions are clear
Objectivity	the extent to which data is unbiased, unprejudiced, and impartial
Relevancy	the extent to which data is applicable and helpful for the task at hand
Reputation	the extent to which data is highly regarded in terms of its source or content
Security	the extent to which access to data is restricted appropriately to maintain its security
Timeliness	the extent to which the data is sufficiently up-to-date for the task of hand
Understandability	the extent to which data is easily comprehended
Value-Added	the extent to which data is beneficial and provides advantages from its use

2.2.2 Cryptographic-Based Trust. Cryptographic trust is seen in everyday browsing of the World Wide Web (WWW), as secure browser traffic is encrypted and transmitted via SSL. Technology is used to give users a reasonable assurance that data they have entered is safe from prying eyes. SSL is also serves as a guarantee to the user that the site they are visiting is really the site it purports to be. Kearney *et al* propose that cryptography serves three purposes: to maintain confidentiality, to provide a means of authentication, and to provide a means of verifying integrity for a piece of information [20].

Confidentiality prevents those without a private key from accessing the information, allowing only intended recipients to view the information. Authentication also requires a private key to gain access, and is combined with the integrity verification in a digital signature. The signature

"provides a means of detecting whether the document has been altered" [20]. Vital to this concept is that an information recipient must be able to reliably determine whether the signature certificate key is correct. This assurance is provided by signing the key and can be done one of two ways. Kearney *et al* state: "the certificate can be 'self-signed', in which case it offers little assurance about the 'true' identity of its owner, but can still be useful in checking that one is still 'talking to the same person as before'. Alternatively it can be signed by an 'authority'. In this case, the confidence in the identity information given in the certificate depends on the trust that can be placed in the authority and the processes that authority uses to validate information before issuing a certificate" [20].

2.2.3 The Trust Management Philosophy. In their article from the Summer 1997 issue of *World Wide Web Journal*, Rohit Khare and Adam Rifkin present the idea that to focus exclusively on cryptographic means of security is too narrow of a focus [21]. They present several scenarios and demonstrate that there are several unknowns despite the cryptographic protections afforded to users. Consider the situation where a customer connects to her banks' website and initiates a payment to her landlord. The bank then has to decide if they should indeed transfer the money from the customer to the landlord. Even if we make the assumption that the transaction had appropriate cryptographic protections enforced, there are many details that are still not known. For example, "nothing in the online encryption handshake necessarily establishes [the customers] identity, no recoverable signed document exists as a receipt for the transaction, [and] no verification exists that the landlord toted up the rent correctly" [21]. The answers to these and other questions cannot be found in the cryptographic realm; additional measures must be put in place to ensure complete trustability.

There are three basic principles that drive the Trust Management philosophy: *be specific*, *trust no one but yourself*, and *be careful*. While seemingly obvious, *being specific* equates to leaving no detail untouched. In our banking example it is not enough for the customer and bank to agree to trust each other; minute details are required. If a direct deposit is set up with a bank, there is an

agreement that "specifies precisely the account into which the employer will forward funds, the times payment will be made, and the expected bank recourse in cases of error" [21]. Once this agreement is in place, both the customer and bank can safely agree to trust one another within the bounds of the specified contract, but transactions outside the bounds of the agreement should not be trusted.

While it sounds exclusionary in nature, the principle of *trust no one but yourself* works in the opposite manner. Its basic tenet is that "any trust decision should logically be derived from the axioms you yourself believe" [21]. Considering the example of believing that your credit card number is *1234*, one works backwards until a chain of trust is established. What you *really* know, is that *1234* is the credit card number that a credit company issued you, and you believe that they are a valid credit company because your bank says so, and you trust your bank because its public key is the same one they presented when you initially opened your account with them. At that point it can safely be said that your credit card number is indeed *1234*.

Despite the best intentions and with the best security mechanisms in place, one must always remember to *be careful*. There is no scientific methodology associated with this principle, it is merely a calling to verify and logically prove every decision made, taking time to fully study the ramifications of a decision. One slip-up is all that is needed to defeat even the most rigorous security methodology. System designs must be scrutinized at every step of the way, then checked and rechecked to ensure that no trustability compromises have been introduced.

*2.3 I*I Framework*

As a reminder to the reader, I*I is defined as "the correctness of information, includ[ing] the accuracy, consistency, and reliability of information domains (content, process, & system) of an enterprise" [13]. To that end it defines three domains: *content*, *system*, and *process*. Each domain contains three attributes: *accuracy*, *consistency*, and *reliability*. Together the domains and their

attributes are used to calculate an overall value for the I*I of a particular set of information. The *content* domain is the set of actual data provided to users for consumption. Content can come in many forms including text, graphics, audio, or video. The *process* domain provides a set of functions that transform an input into a specified output. Finally, the *system* domain is the set of components (both physical and logical) that are configured for a certain purpose. Examples of a system include computer applications as well as business organizational units. Figure 2.3 illustrates the domains with their attributes.

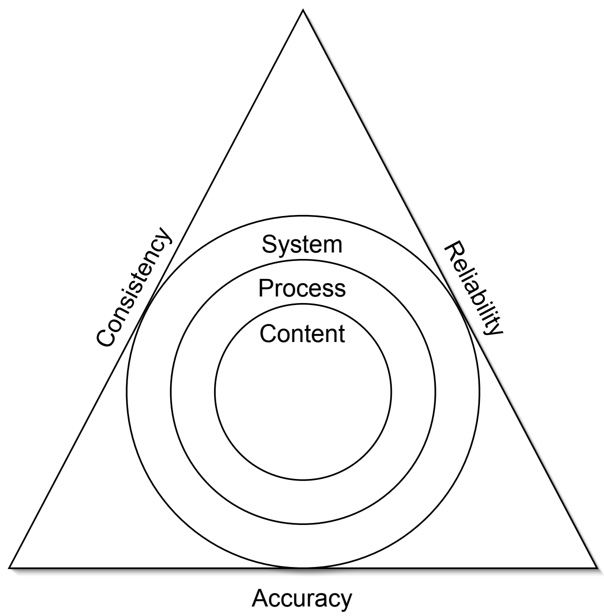

Figure 2.3: Information Integrity Domains [22]

When considering the attributes, Nayar states that in general they "apply to each of the domains ... and can be objectively evaluated and measured" [22]. *Accuracy* is assessed by comparing data to an established standard and setting an allowable tolerance of deviations from the standard. If information falls within that tolerance, then it is considered accurate. *Consistency* is evaluated from repetitive instances of the same data occurring "in space, over time, and in relation to one another at the same point in time" [22]. It is expected that the same inputs, assumptions, and conditions will produce the same results time after time. When this does not happen, confidence

in the information is lowered. *Reliability* is determined by comparing the completeness of information "when compared to a given specification; by assessing its currency or relative newness; and by establishing its verifiability" [22]. Each attribute with its respective elements is shown in Table 2.2.

Table 2.2: The Attributes of I*I With Respective Elements [22]

Accuracy	Consistency	Reliability
Standards Tolerance	Spatial Temporal Relational	Completeness Currency Verifiability

With the domains and attributes defined, the I*I Framework proposes a set of formulae with which to calculate the I*I for a given set of information. Each domain uses its own formula, and the results from each domain are multiplied together for a final result. The input values for each domain formula are between 0 and 1. Letting A^x represent the attribute *accuracy* for each domain x, C^x represent the attribute *consistency* for each domain x, and R^x the attribute *reliability* for each domain x, the individual formulae are shown in the following equations, with the final I*I formula being given in Equation 2.4.

$$\mathrm{I{*}I}^{Content} = A^C \times C^C \times R^C \tag{2.1}$$

$$\mathrm{I{*}I}^{Process} = A^P \times C^P \times R^P \tag{2.2}$$

$$\mathrm{I{*}I}^{System} = A^S \times C^S \times R^S \tag{2.3}$$

$$\mathrm{I{*}I}^{\mathfrak{I}} = \mathrm{I{*}I}^{Content} \times \mathrm{I{*}I}^{Process} \times \mathrm{I{*}I}^{System} \tag{2.4}$$

2.3.1 Example Computations. Using these equations, we can devise a multitude of ways to arrive at the same value for $\mathrm{I{*}I}^{\mathfrak{I}}$, as demonstrated in Table 2.3. The shaded cells denote input values that deviate from 1.00. This leaves one of two possibilities: either we accept the ambiguity of the final value by itself and hope that it will be enough to guide our decision making process,

or we can retain the complete set of inputs that generated the value. The I*I framework does not provide an easy way to retain granular meaning for a particular trust value, nor does it provide any way to reconstruct a value without the original set of inputs.

Table 2.3: I*I Framework Example Computations

Attributes	Accuracy	Consistency	Reliability	TOTAL
Domains:				
Content	1.00	0.20	1.00	0.20
Process	1.00	1.00	0.70	0.70
System	1.00	1.00	1.00	1.00
TOTAL	1.00	0.20	0.70	0.14

Attributes	Accuracy	Consistency	Reliability	TOTAL
Domains:				
Content	1.00	1.00	1.00	1.00
Process	0.45	1.00	0.40	0.18
System	0.80	1.00	1.00	0.80
TOTAL	0.36	1.00	0.40	0.14

Attributes	Accuracy	Consistency	Reliability	TOTAL
Domains:				
Content	1.00	1.00	0.14	0.14
Process	1.00	1.00	1.00	1.00
System	1.00	1.00	1.00	1.00
TOTAL	1.00	1.00	0.14	0.14

Attributes	Accuracy	Consistency	Reliability	TOTAL
Domains:				
Content	0.80	0.90	0.86	0.62
Process	0.43	0.97	0.96	0.40
System	0.89	0.67	0.95	0.57
TOTAL	0.31	0.58	0.78	0.14

2.4 Summary

This chapter has discussed the current state of SOAP web services, addressing the security void as well as current proposals to implement security. Two types of trust: 1) data quality trust, and 2) cryptographic trust were discussed, as well as the Trust Management Philosophy which borrows concepts from both types of trust. Finally, the I*I Framework was described in detail, culminating in the equations for determining I*I for a given set of information. Example computations were given to demonstrate a potential weakness of the framework. All of this sets the stage for the next chapter, where the proposed trustability assessment model is given in full detail.

III. Methodology

Many of the concepts outlined in Section 1.2 are given in detail within this chapter. Discussion is first given on how trustability of information should be represented. Second, a realistic, useful set of indicators are defined with which to determine trustability. Each indicator is comprised of a set of attributes, which are laid out next. Once the trust has been calculated, it is encoded, giving consideration to previous discussion on trustability representation. Key to the usefulness of trust values is their interpretation, and discussion is given to this. Finally, everything is framed within a web services context, setting the stage for an implementation using the WS-Security specification.

3.1 Overview

Figure 3.1 provides a birds-eye view of the methodology. Each transaction that occurs in the web service results in calculating a trust assessment of the information from the transaction. The trust assessments are stored in a transaction log as hexadecimal values for later interpretation. The trust value of `CFAE10` in the figure is an example of a calculated trust assessment. When the decision maker requests information from the web service, the trust assessments are presented along with the data. Each assessment is interpreted, or translated, to a decimal percentage and displayed with color coding as explained in Section 3.2. Users are given the opportunity to adjust the amount of tolerance for reduced trustability, allowing for the best interpretation for the decision being made.

3.2 Representation Considerations

An overall goal of an information retrieval system should be to present relevant, useful *data* to decision makers. Accordingly, it follows that any representation of *trust* for a given data set should be presented in a relevant, useful manner that lends itself to the decision making process. Intuitively this implies that a simple pass or fail model is not adequate. We are looking for degrees

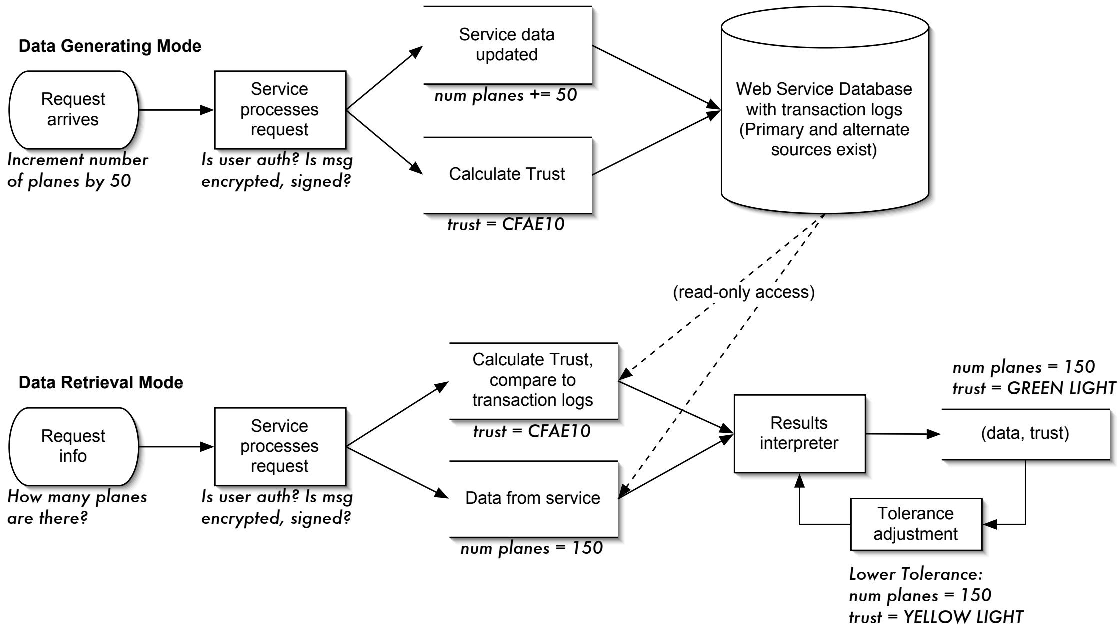

Figure 3.1: Information cycle in the secure web services environment

of trustability that a decision maker can combine with their human intelligence and personal experiences to make the best decision. While a pass/fail model is not adequate, the proposed model is still comprised of a discrete set of values, providing other systems the ability make decisions based on the trustability of information.

Along with the actual representation of trust, it is important to discuss when in the information flow process the trust data is presented to users. For the sake of this work, we will consider our environment in two modes: a data generating mode and a data retrieving mode, which are illustrated in Figure 3.1. The data generating mode operates in a continuous cycle, regardless of any activity in the data retrieval mode. Trust values are calculated and stored in this mode as services are accessed, with various indicators giving visibility into the trustability of information available in this mode. Trust calculation involves harvesting each indicator value and building a composite assessment value that is stored for later interpretation. Each assessment value is stored in a cumulative transaction log that can be later mined for historical trends.

The data retrieval mode is a read-only environment used as needed by the decision maker. When information is requested, the previously calculated trust values are also retrieved and their interpretations are displayed along side of the requested data. Trust interpretations are presented as a roll-up of indicator trust values with the composite assessment being conveyed visually with color coding. The color display is divided into three user defined categories from low to high represented by red, yellow, and green respectively. Individual indicators are also presented with this visual aid to alert users if any specific indicators have significantly affected the trust value. Users are able to obtain detailed values for each indicator to see why a particular value was assigned, allowing the best decision to be made for the situation at hand. Also presented is the ability to make adjustments to the amount of tolerance allowed (see Section 3.5.1 for a detailed discussion of allowable tolerance), resulting in altered interpretations specific to the decision being made.

3.3 Indicator Definitions

For the purposes of this work, trustability indicators must be able to be readily gathered from a secure web services environment. Recognizing that this represents but a subset of all possible indicators that can be used to determine trustability, our focus is limited to that pertaining to web services. Indicators are either directly gathered from the WS-Security framework, or are able to be easily determined from the information system as a whole. Nebulous indicators requiring subjective input have not been considered for this work. Six indicators have been identified and are outlined and described in the following sections. They are WS-Security Authorization, WS-Security Signature, WS-Signature Encryption, Data Consistency, Source Credibility, and Data Currency.

3.3.1 WS-Security Authorization. This indicator is the first of three from the WS-Security family. These are among the most pivotal indicators, as they deal with the security of the infor-

mation system. Information retrieved from an unsecured system should not be trusted because there are no assurances as to who or what has had access to the data. Vital to our determination of trustability is a secure system, one where we know and trust anyone who has read or modified any data. In addition, we need to know if they accessed the system in a manner that will not compromise its security.

WS-Security Authorization measures whether or not users accessing the system are authorized. In the event that an unauthorized user is able to gain access to the system, they could conduct any one of a number of malicious actions. Data values could be altered or deleted, information about non-existent entities could be entered, or they could simply access information intended only for internal use. In any of these cases this indicator would lower the trust value. However, access by authorized users is assumed to be trusted for the purposes of this indicator, and would result in a higher trust value.

3.3.2 WS-Security Signature. Assuming a user is authorized to access the information system, they must provide some guarantee that their identity is valid. Signature mechanisms provide these assurances, and this indicator gauges the process and mechanisms used. If the message was properly signed with authorized keys, then the indicator raises the trust value. Conversely, an incorrectly signed or unsigned message lowers the trust value.

3.3.3 WS-Security Encryption. Given that a user is authorized, and has signed the message asserting that they are really the ones accessing the service, the next item of concern is whether or not anyone else is able to see the message traffic. Message level encryption provides an assurance that the contents of the message are viewable only to the intended recipients. This indicator is a measure of whether or not encryption was properly and securely applied to message traffic. A message that has had encryption correctly applied raises the trust value, whereas one with missing or incorrect encryption results in lowering the trust value.

3.3.4 Data Consistency. Figure 3.1 illustrates an environment that exists with multiple instances, all intended to provide the same information to different users at different locations. In this case, it is useful to know whether data being retrieved from a particular instance exists within other instances. Data consistency is the measure of data retrieved compared to similar data that exists in alternate sources. Whether similar data exists in one, many, or no alternate sources is reflected in the trust value. Obviously the more sources that contain similar data, the higher the trust value, whereas fewer or no alternate sources result in a lower trust value.

3.3.5 Source Credibility. Consistent with the notion of multiple sources providing the data, each source has a reputation associated with how accurate and trustable their information is. If a particular source has a reputation for providing inaccurate information, then future transactions will result in a lower trust value. However a source that is considered to be highly accurate and consistent will result in a higher trust value.

3.3.6 Data Currency. Information often has a certain life expectancy, after which is considered to be stale, or outdated. Decisions made based on stale data could have adverse effects on the mission at hand. The goal of this indicator is not to eliminate stale data from entering the system, but rather alert decision makers to its presence and lower the information trustability. Data that is as current as or newer than the expected currency for its data type will raise the trust, whereas data that is older will lower it.

3.4 Indicator Encoding and Calculation Scheme

In order to store these indicators a model must be devised in which granular meaning can be retained. While the Information Integrity framework discussed in Chapter II stores trust as decimal values, this model attempts to build and retain *reconstructible* trust values. We can conveniently have each indicator attribute represented by one bit in a binary system, resulting in a four-bit representation for each indicator trust value. To make manipulation of the values easier, we then

convert each binary value into a hexadecimal value, leaving us with a single digit representing trust for *each* indicator. With six indicators, the entire trust value is represented as a six digit hexadecimal number. Given the same set of inputs, this number can be precisely reconstructed over and over again. Similarly it can be reverse engineered every time to determine what set of inputs resulted in its final value.

Figure 3.2 illustrates a trust value and how it is deconstructed and parsed. The entire assessment is represented as `DD7F63`, one digit for each indicator. Each indicator value can be parsed as a binary value, illustrated with the first digit of `D` which is represented as `1101` in binary. As discussed in the following sections, attribute bits are paired together when the trust value is parsed for meaning. For now it is enough to say that the first two bits of `11` represent an unflawed attribute. The last two bits of `01` represent the assessment impact bounds discussed in Section 3.4.1, interpreted as a medium-narrow impact bracket. Contribution of the attribute and impact bracket values to the overall interpretation are discussed in Section 3.5.2. As is discussed in Section 3.4.1, the Data Currency indicator is represented without bracket preferences. Parsing a Data Currency indicator results in two attribute groups, the first representing the actual currency of data, and the second representing the expected currency of data. This is discussed with more detail in Section 3.4.5.

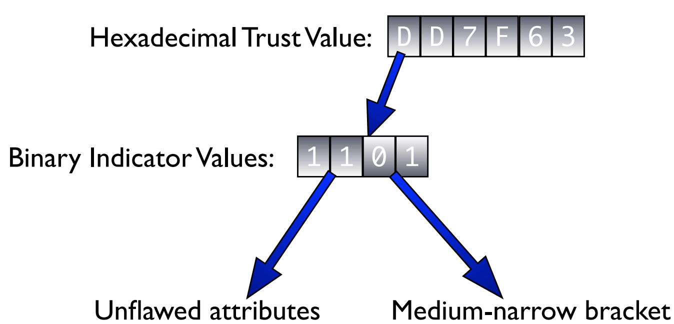

Figure 3.2: Example of indicator encoding

The fact that we have a reconstructible value is critical; it enables us to retain the complete impact of the calculated trust . As an example, consider an indicator with a binary value of `1101`,

or `D` when converted to hexadecimal. Of its four attributes, the first two and last one are positive, while the third attribute/bit is negative. Without knowing specifically what the attributes for this indicator are, we can see that the third attribute is lowering the trust in some way. Personal experience and intelligence of the decision maker comes into play here, they can make appropriate decisions now knowing why the trust has been lowered. Due to the methodical nature of trust calculation, the same situation will always result in this value, and likewise this value will always correspond to the same situation.

3.4.1 Indicator Bracketing. For all indicators except Data Currency, an indicator bracket is part of the trust value. The bracket is used when interpreting the trust; the wider the bracket the broader the range of possible scores a given indicator can contribute to the overall interpretation. Represented with two bits, the brackets range from low to high as shown in Table 3.1 and are specified by the user. Assignment and representation of the brackets are straightforward, but interpretation requires further discussion given in Section 3.5.2.1.

Table 3.1: Attribute bracketing scale

Bracket	Meaning
`00`	Narrow range
`01`	Medium-narrow range
`10`	Medium-wide range
`11`	Wide range

3.4.2 WS-Security Indicators. The encoding for each of the three WS-Security indicators (Authorization, Signature, and Encryption) is identical. These indicators capture whether the WS-Security components were successfully applied or not. This obviously requires only one bit, but in keeping with the four bit representation of other indicators, a leading zero is prepended to the trust value. Thus *each* WS-Security indicator receives a two bit representation to denote whether it was successfully applied or not. Incorporating the predefined two bit bracket scheme results in a four bit representation.

3.4.3 Data Consistency. When a web service transaction takes place, the transaction data is compared to similar data in alternate web service sources. It is assumed that the identification tags are unique for each element of data, and that each source stores a replicate of the data in the same format with the same record fields. If a record exists for an aircraft with the ID equal to `a3` with fields `model` and `status`, then any source that contains data about an aircraft `a3` is referring to the same aircraft data, with fields `model` and `status`. Each alternate source either contains a record for aircraft `a3` or it doesn't. If it does, then the details of the record are compared to that of the primary source. In the general case, a complete match between the primary and alternate sources means that the data is consistent between the two and will yield a high trust value. An incomplete match or non-existent record means that the data is not able to be verified as consistent, which will lower the trust value. The present work examines two alternate sources, although the methodology could be expanded to more sources as needed. Attribute values are set for both sources as shown in Table 3.2.

Table 3.2: Data Consistency Scheme

Value	Meaning
`00`	Both alternate sources are NOT consistent
`01`	First alternate source NOT consistent, Second alternate source IS consistent
`10`	First alternate source IS consistent, Second alternate source NOT consistent
`11`	Both alternate sources ARE consistent

When calculating the data consistency for a particular transaction, there are four types of transactions that may occur: retrieval, addition, updating, or deletion. The basic computation for each case is the same; we distinguish between them because the actual steps taken to compute trust are different for each with the exception of addition and update methods. They are grouped together, leaving us with three ways to calculate the data consistency value. The following discussion assumes method invocation for an aircraft with id `a3`, although the methodology is the same for any data type.

1. **Retrieval methods:** The first alternate source is queried for existing aircraft with an id of `a3`. Any results are checked field by field to see if the record completely matches. In the case of aircraft `a3`, the `model` and `status` fields from each alternate source are compared to those in the primary source. If both fields match, then the attribute bit representing the matching source is set to `1`. If either of the fields do not match, then the bit is set to `0`. If no results are returned, then the record obviously does not match, and the bit is set to `0`.

2. **Addition/update methods:** This process differs from the retrieval methods in that alternate sources are queried for data by *all* fields of an object. If the `model` field of aircraft `a3` is set to `F-16`, and the `status` field is set to `ready`, then each alternate source would be queried for aircraft with an ID of `a3`, a `model` of `F-16`, and a status of `ready`. The returned results are processed the same manner as the retrieval methods. Complete record matches yield an attribute value of `1`, and incomplete matches or empty result sets yield a value of `0`.

   It should be noted that the first time an object is added to the information space, it will not exist in alternate sources, resulting in a Data Consistency value of `0` for each alternate source. Despite this low consistency rating, the newly added data could be accurate. This indicator is not a measure of accuracy, rather its only purpose is to measure whether the data is consistent across multiple sources, regardless of accuracy. Answering the question of accuracy requires a source that has been certified as being accurate. Such a consideration is beyond the scope of this work, but is mentioned to clarify our intentions for this indicator.

3. **Deletion methods:** In this type of method the first alternate source is queried for existing aircraft with an id of `a3` as is done for the retrieval methods. If there are results returned then the alternate source is *not* consistent since the aircraft `a3` is in the process of being deleted from the primary source. Accordingly the attribute bit is set to `0`. An empty result set indicates that the source is consistent and receives an attribute value of `1`. Similar to

the addition/update methods, this does not measure whether deleted data is accurate, only whether it is consistent among other sources.

The last two attribute bits of the Data Consistency indicator are reserved for user bracketing preferences as outlined in Table 3.1. The complete algorithm for determining attribute values as described above is attached to this document in Algorithm A.2.

3.4.4 Source Credibility. In this model, sources are assigned a credibility rating on a scale from one to four, represented in two digit binary, as shown in Table 3.3. These values are assigned initially based upon the likelihood of accurate data coming from a particular source. As the service is accessed by users, each source builds a historical transaction log allowing the credibility ratings to be adjusted. Higher Data Consistency ratings over time will serve to raise the source credibility, whereas a trend of lower consistency will lower the source credibility. Like the previously mentioned indicators, the last two bits of this indicator are a bracket assignment, as outlined in Table 3.1.

Table 3.3: Source Credibility Values

Rating	Meaning
00	Low credibility
01	Medium-low credibility
10	Medium-high credibility
11	High credibility

3.4.5 Data Currency. In order to determine a value for Data Currency all data types are assigned an *expected* currency, based on how often they *should* be updated (explained below). When a message arrives for processing, the current time is captured and compared to the last time that an update occurred for the data object being accessed. The scheme used for representing currency values is shown in Table 3.4. The first two bits of the indicator value represent the actual currency, and the last two represent the expected currency.

Table 3.4: Currency bit representation

Bit Rep	Meaning
`00`	Data is on the magnitude of minutes old
`01`	Data is on the magnitude of hours old
`10`	Data is on the magnitude of days old
`11`	Data is on the magnitude of weeks old

Data that remains fairly static will receive a longer expected currency than a data type that is very dynamic. As an example, Air Force Major Commands (MAJCOM) usually have the same commander for two years; the number of bases associated with a particular MAJCOM does not change very often. Thus, a MAJCOM data type would have the longest *expected* currency value assigned. At the opposite end of the spectrum is information about on-hand munitions inventories. It is assumed that information is updated as munitions are used, so it follows that it would be assigned a short *expected* currency value.

If a method is invoked for a MAJCOM object at timestamp `2005-01-22-1200`, transaction logs will be checked to see when that particular MAJCOM object was last updated. If the timestamp of the last update is `2005-01-20-1200`, then the currency of this message is on the magnitude of days old (exactly two day to be precise), thus the current currency value would be `10`. Combined with the expected value of the MAJCOM type, the resulting trust value would be `1011`, or `B`. Since the actual currency is better than the expected currency, this will raise the trust value of this indicator.

3.5 Interpretation of Encoded Trust

Even with a repeatable method of building trust values, an interpretation of a particular value must be given when information is requested in the Data Retrieval mode shown in Figure 3.1. Six digit hexadecimal numbers convey little in terms of practical interpretation. Machines making decisions based on these generated values can parse each whole value and easily determine why it received the value that it did and act accordingly. Thus, this avenue of interpretation is meant primarily for the human use and interpretation of the trust values, although the use of allowable

tolerance is certainly an important consideration for machine based decisions as well. Before the details of interpretation can be given, we must first discuss the notion of *allowable tolerance* in a decision.

3.5.1 Allowable Tolerance. All decisions carry a certain level of impact with their outcome. At a simplified level, consider the decision of which runway an aircraft should use when taking off. Aside from local ground traffic management policies which may dictate the use of one runway over another, this decision has no impact on the end result of the flight. Once the aircraft is airborne the appropriate course will be set to wherever the destination is, regardless of which runway was used for takeoff. Conversely, the decision of whether or not to refuel the aircraft has a significant impact on the ability to reach the final destination; lack of fuel significantly reduces the range of the aircraft. Similar impacts exist within decisions made on the battlefield, and a decision support system such as the one proposed should have allowances for adjusting the importance of the decision being made.

Expressing this adjustment as "dial-a-trust", each indicator is given an adjustable tolerance dial that can be raised and lowered. A raised tolerance dial indicates a high tolerance, or a wide range of tolerance in the decision at hand. The general effect of raising the tolerance dial will result in a higher trust interpretation, giving the user more confidence in the data at hand. Naturally it follows that a lowered tolerance dial narrows the range of tolerance, effectively lowering the trust interpretation. This should bring a certain apprehension to the decision maker as certain thresholds are crossed.

In the case where a particular trust value is strongly in the "green" zone of the display, adjustments to the allowable tolerance would not likely be seen. However, considering the case where an initial interpretation results in a value towards the bottom of the "green" zone, a lowered tolerance could easily bump the interpretation down into the "yellow" zone. This case serves to alert the de-

cision maker to the weakness of the initial interpretation. Similarly, if an interpretation is towards the top of the "yellow" zone, a raised tolerance could easily translate to a green representation.

In our simplified example, when deciding which runway to use the tolerance dial could be raised. This would in effect raise the trust interpretation of the data at hand, allowing a free decision to be made from any available runway. When deciding whether or not to refuel, the tolerance dial would be lowered, putting focus on the importance of refueling. Decision makers would hopefully decide to refuel the aircraft to allow the destination to be safely reached.

The present work proposes five levels of tolerance: low, medium-low, normal, medium-high, and high. The contribution of each level to the interpretation value us discussed in the following section. In the general case, a low tolerance lowers the interpretation, while a high tolerance raises the interpretation.

3.5.2 Interpretation Methodology. As mentioned earlier, interpretation of trust values is primarily intended for human users. In this light it can be thought of as a translation of the hexadecimal numbers to a decimal percentage. Each of the three WS-Security indicators are interpreted in the same exact manner, while the other indicators are all interpreted uniquely. Each indicator receives a value between 1 and 100 representing a percentage of how trustable the information is thought to be.

3.5.2.1 WS-Security, Source Credibility, and Data Consistency Indicators. WS-Security, Source Credibility, and Data Consistency indicators are grouped together for this discussion because the trust representation for each incorporates bracketing preferences. The first two bits of the trust value are the actual attributes used to gauge trust, and the second two bits designate the bracketing placed on that indicator. Each two-bit attribute group is given a high and low percentage bound, within which the interpretation value must fall. The percentage bounds for each indicator are shown in Table 3.5. The range between these bounds is divided into 20 values

grouped into four brackets. Within each bracket are five values, each value representing the trust interpretation according to the amount of allowable tolerance.

Table 3.5: High and Low Percentage Bounds

Indicator	Attribute	High	Low
WS-Security	00	30	1
WS-Security	01	**31**	**100**
Source Credibility	00	30	1
Source Credibility	01	50	31
Source Credibility	10	70	51
Source Credibility	11	**71**	**100**
Data Consistency	00	40	1
Data Consistency	01	60	41
Data Consistency	10	60	41
Data Consistency	11	**61**	**100**

It should be noticed that the unflawed attributes (01 for WS-Security; 11 for Source Credibility and Data Consistency) have reversed high and low percentage bounds. This is designed on purpose so that the desired bracketing results in an accurate interpretation value. When attributes are equal to anything other than 11, something took place to lower the trust of that indicator and we will consider the indicators' input to be flawed. Setting aside the indicator bracket, if a Data Consistency indicator has an attribute value of 10, then according to Table 3.2 the second alternate source did *not* have consistent data. In this case, the "perfect" scenario occurs when both sources are consistent, but since only one of them was deemed consistent, the situational input is flawed.

Each interpretation value is calculated using a *multiplier* value. Table 3.6 lists the multiplier values for each bracket and tolerance pair for flawed attribute groups (00 for WS-Security; 00, 01, and 10 for Source Credibility and Data Consistency). When the attribute value is not flawed (01 for WS-Security; 11 for Source Credibility and Data Consistency), the multiplier is determined differently since the percentage bounds are reversed. For example in the case of a WS-Security indicator with attribute value 01 the "high" value is 31 and the "low" value is 100 (from Table 3.5). The multiplier values for this case are shown in Table 3.7. The multiplier values ensure that the interpretation values increase within each bracket, yet the overall interpretation values decrease

for flawed attributes and increase for flawless attributes. In this way a flawless attribute has greater potential to *increase* the interpretation, while a flawed attribute has greater potential to *decrease* the interpretation.

Table 3.6: Multiplier Values for Flawed Attribute Groups

Bracket	Tolerance	Multiplier
00	Low	15
00	Medium-low	16
00	Normal	17
00	Medium-high	18
00	High	19
01	Low	10
01	Medium-low	11
01	Normal	12
01	Medium-high	13
01	High	14
10	Low	5
10	Medium-low	6
10	Normal	7
10	Medium-high	8
10	High	9
11	Low	0
11	Medium-low	1
11	Normal	2
11	Medium-high	3
11	High	4

With the high and low percentage bounds from Table 3.5 designated as percentHi and percentLo respectively, and a multiplier from Tables 3.6 and 3.7, the formula used in the general case for calculating the interpretation is shown below in Equation 3.1. Figures 3.3 - 3.5 illustrate the possible interpretations for each indicator type using this formula.

$$\left(\frac{\text{percentHi - percentLo}}{19} \times \text{multiplier}\right) + \text{percentLo} \tag{3.1}$$

Interpretation for each of the indicators types discussed here is calculated using the same formula. The difference lies in the percentage bounds specified in Table 3.5, and the number of attribute groups. WS-Security only uses one bit to indicate its attributes, which when prepended with a leading zero only leaves two attribute groups. Source Credibility uses both attribute bits

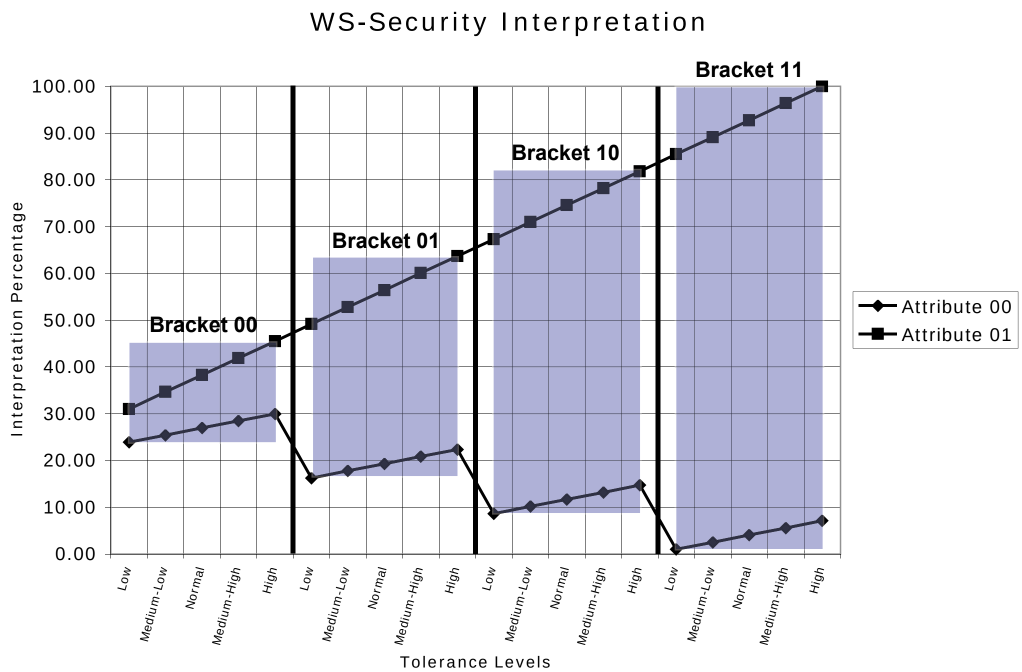

Figure 3.3: WS-Security Indicator Interpretation Chart

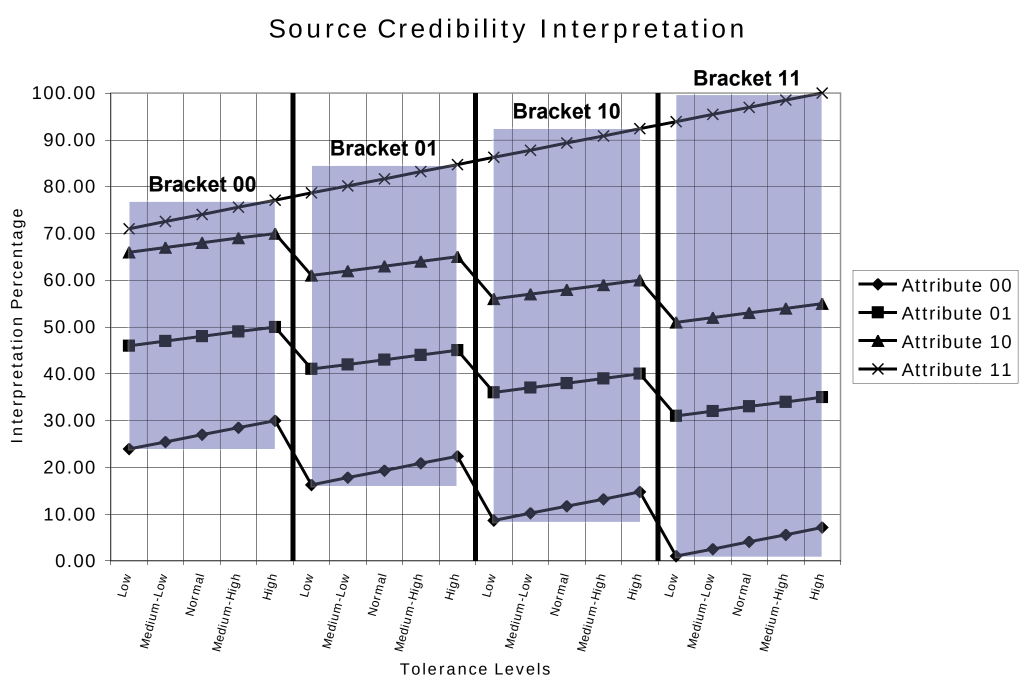

Figure 3.4: Source Credibility Indicator Interpretation Chart

Table 3.7: Multiplier Values for Flawless Attribute Groups

Bracket	Tolerance	Multiplier
00	Low	19
00	Medium-low	18
00	Normal	17
00	Medium-high	16
00	High	15
01	Low	14
01	Medium-low	13
01	Normal	12
01	Medium-high	11
01	High	10
10	Low	9
10	Medium-low	8
10	Normal	7
10	Medium-high	6
10	High	5
11	Low	4
11	Medium-low	3
11	Normal	2
11	Medium-high	1
11	High	0

to represent its trust value, and so has four attribute groups. Data Consistency also uses both attribute bits, but its nature requires a unique approach. Since the attributes `01` and `10` represent one of two alternate sources being consistent, there is no distinction in the contribution to trust value that either source brings. Thus, they receive the same set of interpretation values using the same percentage bounds.

As an example, consider a WS-Security indicator with the value `0010`, and a normal tolerance. From Table 3.5 we set percentHi equal to 30, and percentLo equal to 1. Looking up the appropriate multiplier from Table 3.6, we set the multiplier equal to 7. The resulting trust interpretation for that indicator is 11.684%, as shown in Equation 3.2.

$$\left(\frac{30.000 - 1.000}{19} \times 7\right) + 1 = 11.684 \tag{3.2}$$

Alternatively, if we have a Data Consistency indicator with value `1101`, and a high tolerance, we must use a multiplier for the case when all attributes are not flawed. We set percentHi equal to

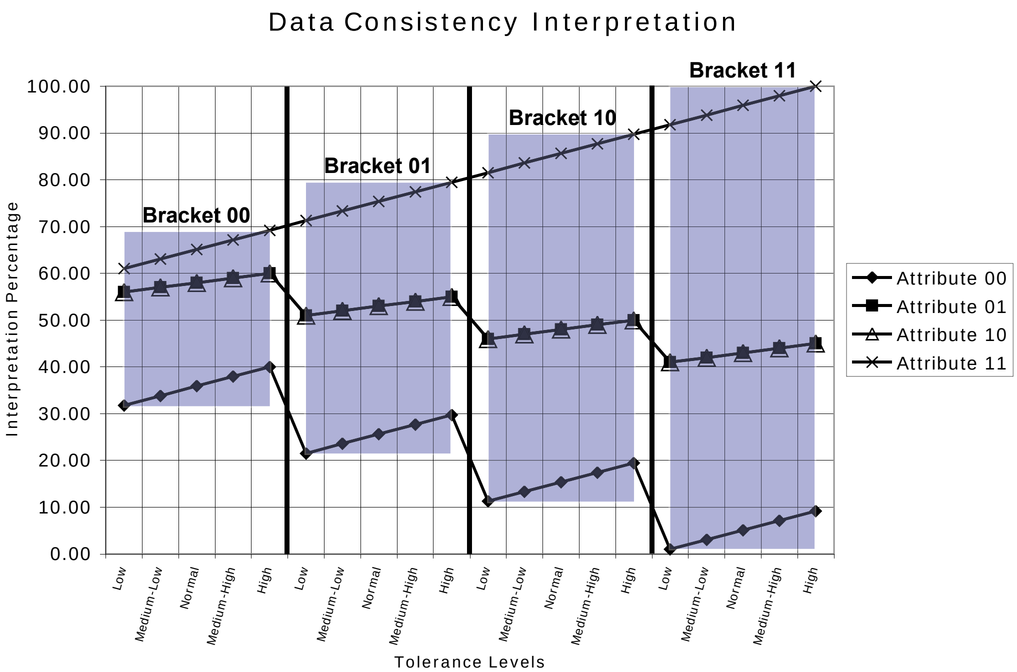

Figure 3.5: Data Consistency Indicator Interpretation Chart

61, percentLo equal to 100, and set our multiplier equal to 10 from Table 3.7. Our trust interpretation is calculated as 79.474% as shown in Equation 3.3.

$$\left(\frac{61.000 - 100.000}{19} \times 10\right) + 100.000 = 79.474 \tag{3.3}$$

3.5.2.2 Data Currency Indicator. Calculating the interpretation for Data Currency is much simpler, as there is no tolerance to account for. The general case is that if the actual currency is equal to or newer than the expected currency, then the interpretation will be 100%. Accordingly, an actual currency of `00` automatically receives a 100%. The older the actual currency is when compared to the expected currency the lower the interpretation becomes as shown in Table 3.8. The algorithm for determining this is shown in Algorithm A.1.

3.5.2.3 Interpretation of All Indicators. As one of our stated goals is to provide insight into the overall trustability of the message, we must move beyond interpreting individual

Table 3.8: Data Currency Indicator Interpretation Table

	Expected			
Actual	00	01	10	11
00	100.000	100.000	100.000	100.000
01	66.667	100.000	100.000	100.000
10	33.333	66.667	100.000	100.000
11	0.000	33.333	66.667	100.000

indicators. Interpretation of each indicator results in a percentage, allowing us to easily average the six indicator interpretations to obtain an overall interpretation. Due to the reconstructible trust values the average value does *not* weaken the trust assessment. It is merely an attempt to provide an intuitive understanding at a glance of the trust assessment. The full trust value, as well as individual indicator interpretations are available and shown to the user alongside the overall average.

3.5.3 Web Services Context. As mentioned earlier, the focus of this work is in a secure web services environment, although the concepts discussed thus far could easily take place in any distributed computing environment. Using SOAP-based web services allows us to easily compute the trust for each transaction in the system. As is demonstrated in Chapter IV, there are checkpoints that all incoming and outgoing SOAP messages must pass through. This centralization provides assurance that all transactions are able to be accounted for when determining trust. In other words, there are not any back doors into the system with which one could bypass implemented trust mechanisms. This assumes that direct access to the database is secured appropriately and unauthorized database access is not a factor.

3.6 Summary

This chapter presents a methodology for assessing information trustability in a secure web services environment. Each of the six trustability indicators with their respective attributes are defined. Algorithms are defined with which to calculate a repeatable trust assessment value for

each indicator. Finally, a means of interpreting the derived trust values is given. The presented algorithms and methodologies are implemented in Chapter IV.

IV. Implementation

An example web service has been built to illustrate the use of the trust determination algorithms presented in this work. The service is called AssetTracker, and although the concepts and techniques presented are broadly applicable, we have chosen the application domain of tracking battlefield assets. Sun Microsystems Java™ [4] programming language was chosen for this task, since its cross-platform capabilities make any future enhancements to this work easily accommodated. Additionally, extensive support for web service technologies exist on the Java platform, to include SOAP, WSDL, and XML.

4.1 AssetTracker Overview

The AssetTracker system was designed with several goals in mind. First and foremost, it had to be a web service with methods accessible via SOAP calls. Second, it needed to be a *secure* web service application, specifically using the WS-Security specification requiring user authorization, as well as message level signature and encryption. These first two requirements were driven by the main goal of this work to assess information trustability in a secure web services environment. Thirdly, it needed to simulate information flows and data that may be tracked in a battlefield situation in order to provide some real world applicability. Finally, it needed to easily allow for trust values to be calculated as messages came into the system because of our overarching goal of assessing information trustability.

AssetTracker contains information about combat assets and their locations. At the highest level are location objects consisting of MAJCOM's, which are in turn composed of bases. Each base has a group of assets that are assigned to it. For the purposes of this experiment, assets are limited to aircraft, munitions, and personnel. Individual assets are not associated with each other in any way other than belonging to the same base. The objects and their associations are depicted in Figure 4.1. It is recognized that the real world value of this system falls far below

actual requirements for tracking battlefield assets, however our focus is not the actual application, but rather insight that can be gained into the trustability of its information.

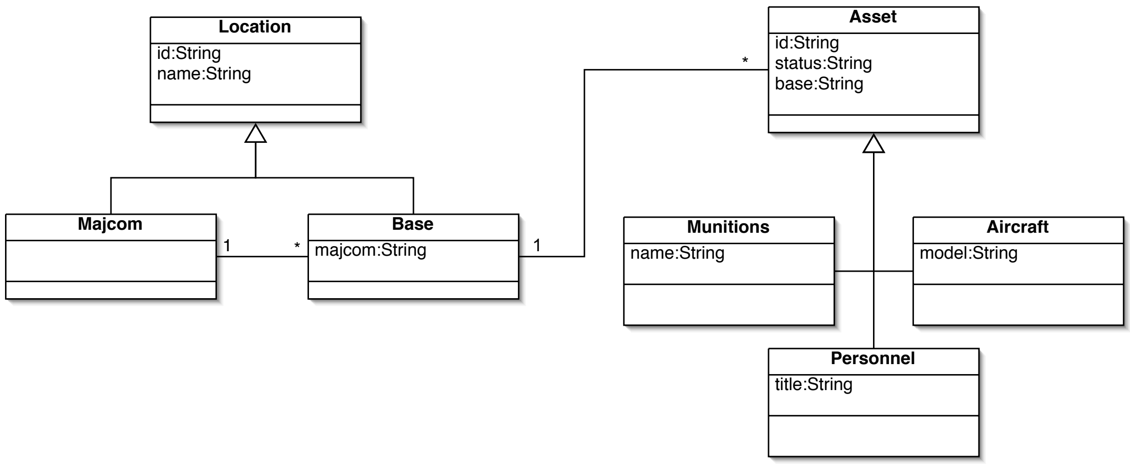

Figure 4.1: UML Diagram for AssetTracker Objects

All AssetTracker objects are stored persistently in a relational database, with appropriate relationships such that it mimics an object oriented programming methodology. This approach was chosen over creating an object hierarchy for reasons of implementation simplicity. Again the goal of this work is to examine information trustability. A partial database schema is shown in Figure 4.2, and a full SQL DDL description is listed in Appendix B. It should be noted that there are additional database tables shown in Figure 4.5 that deal with assessing information trustability. The tables shown here in Figure 4.2 deal exclusively with the AssetTracker objects.

Location and asset objects each have four types of methods: addition, retrieval, update, and deletion. Addition methods can also be thought of as creation methods; it is with these methods that new objects are put into the database. An addition method takes all attributes of the object to add as input parameters. The retrieval methods, using Javabean terminology, are "getter" methods and require only the id of the desired object as an input parameter. A comma delimited string of object attributes is returned by the method. Update methods are analogous to "setter" methods,

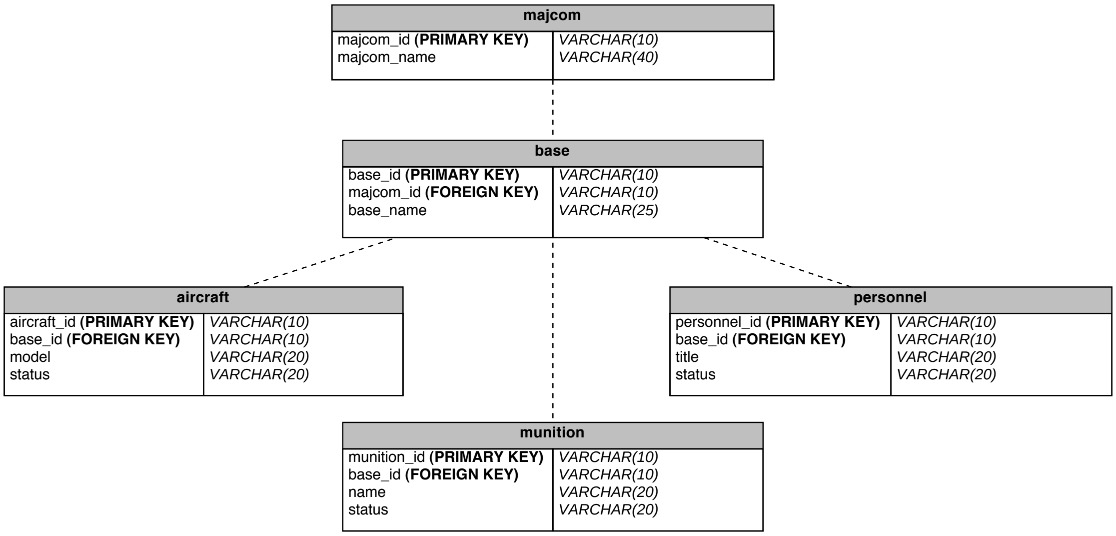

Figure 4.2: AssetTracker Objects: Database Schema

and take as input parameters the id of the object to update, along with the complete list of object attributes with their new values.

Deletion methods accept the object id as an input parameter, but they require some checking before deleting the specified object. Asset objects may be freely deleted since they have no child objects. Location objects however must not have any instantiated child objects in order for them to be deleted. For example, a MAJCOM cannot be deleted unless there are no bases associated with it. As well, a base cannot be deleted unless there are no assets associated with it. These restrictions address standard relational database referential integrity concerns.

The entire AssetTracker system (to include the web service environment) is encapsulated in a set of Java packages organized by functionality. Table 4.1 outlines the various packages and their purpose.

4.2 Web Service Environment

The AssetTracker system is written as a Java class and converted to a SOAP-based web service courtesy of utilities provided with Systinet WASP Server for Java [7]. The AssetTracker

Table 4.1: AssetTracker Java Packages

Package Name	Purpose
`edu.afit.assetTracker`	Web Service code for all three instances
`edu.afit.assetTracker.client.forward`	Client code for the ForwardWS instance
`edu.afit.assetTracker.client.forward.iface`	Interface code for the ForwardWS instance client; generated by WASP utilities
`edu.afit.assetTracker.client.home`	Client code for the HomeWS instance
`edu.afit.assetTracker.client.home.iface`	Interface code for the HomeWS instance client; generated by WASP utilities
`edu.afit.assetTracker.client.rear`	Client code for the RearWS instance
`edu.afit.assetTracker.client.rear.iface`	Interface code for the RearWS instance client; generated by WASP utilities
`edu.afit.assetTracker.persistence`	Manages all database connectivity
`edu.afit.assetTracker.ui`	Main user interface used to generate all sample output
`edu.afit.assetTracker.util`	Interceptor, Handler, and IncomingValidator code
`edu.afit.assetTracker.trust`	Trust interpretation

class is extended to three separate classes which in turn are converted to individual web services. Each one is identical in operation, differing only in the content and battlefield-related location that they represent. This lends itself to a more realistic simulation of battlefield asset tracking, and allows for data consistency checks. The three web services are deployed as *ForwardWS*, *RearWS*, and *HomeWS*.

The *ForwardWS* represents assets of a forward deployed unit, closest to the action on the battlefield. *RearWS* represents a rear supporting unit, slightly distanced from the action, yet still in the battle theater. *HomeWS* represents support from the CONUS, very much removed from the action and theater. Each web service instance has a respective client that represents users of each service. Figure 4.3 shows how each instance interacts with its client.

Each service instance has inherent attributes due to the location it represents. The *ForwardWS* is considered the most current source, being at the center of the battlefield means that it generates information first. The accuracy is assumed to be slightly lower; due to the fast paced nature of its location there is no time to double check information as it is entered into the system. Since *RearWS* is located farther from the battlefield center, it takes longer to receive information updates, lowering its currency. However, that extra time allows for information to be verified thus improving its

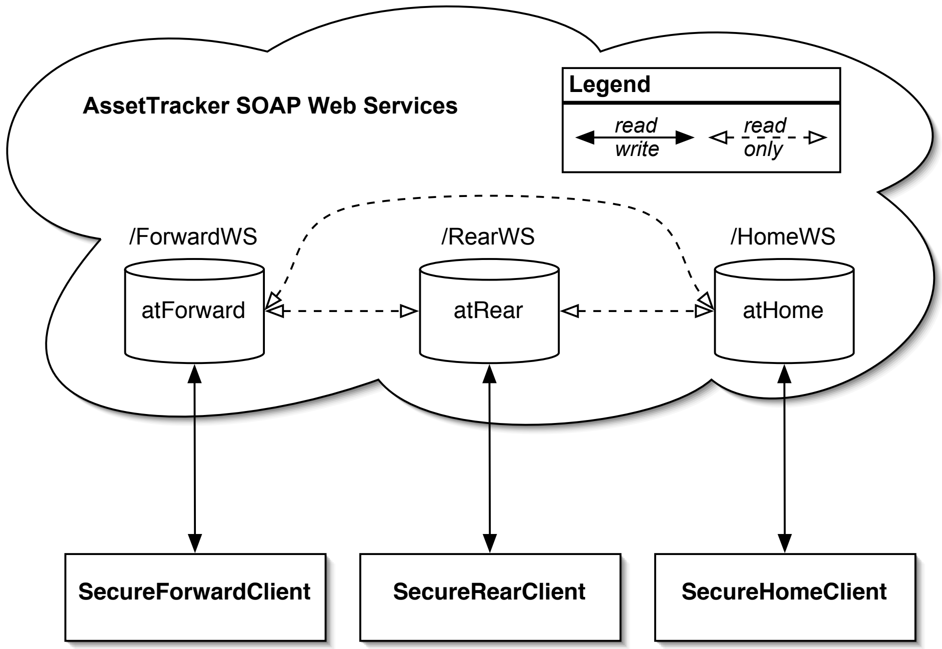

Figure 4.3: Service Instance and Client Layout

accuracy. *HomeWS* is the furthest location, so its currency is the lowest of the three instances. Its distance from the battlefield is so great that its accuracy is considered lower than that of *ForwardWS*; errors caught and fixed at *RearWS* could be re-introduced by the time the information flows back to CONUS.

Functionality of the service clients is straightforward. They each establish a connection to their respective web service, authenticating as the user known to that service. Once connected they are able to invoke available methods, however they provide no user interface as implemented. For illustrative purposes, a user interface to the ForwardWS client is provided in the `edu.afit.assetTracker.ui.AssetTrackerUI` class; no interfaces for the other clients are provided. A variety of methods are invoked in the provided client classes in order to demonstrate a wide range of trust values and outcomes. Actual output from client execution is shown in Section 4.6.

WASP provides several mechanisms for securing message traffic; we are using its WS-Security implementation. All of the methods for each data type are specified to use full WS-Security, i.e. user authorization, signature, and encryption are required for all message transactions. Any message that does not meet these requirements is automatically rejected by the WASP server. The security requirements are specified in a *deployment descriptor* file, where methods are listed with their required security parameters. Methods that are *not* listed in this file do not have any required security; the effect of unsecured messages on trust calculations is discussed in Section 4.3.

When the specified security requirements are met, the message travels through a series of checkpoints before reaching the actual method being invoked. Similar checks are applied on the way back to the client. A graphical representation of this flow is shown in Figure 4.4. The checkpoints are extendible classes provided by the Systinet WASP server and are used to calculate the trust assessment values; we discuss them briefly here and in more detail in Section 4.3.

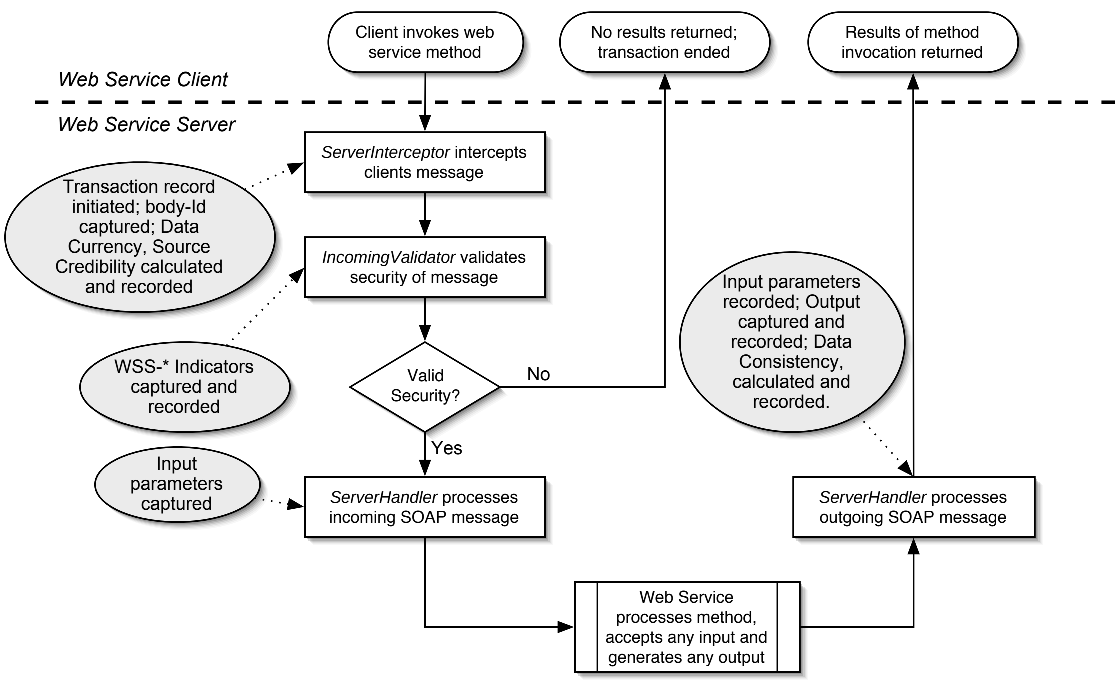

Figure 4.4: SOAP Message Flow

The first checkpoint the message passes through is called a ServerInterceptor. The ServerInterceptor class has access to the raw data in the message; it sees the plain-text XML document. If message-level encryption has successfully been applied, then no data will be able to accessed at this stage. Despite this, certain attributes of an encrypted message are left in clear-text and are able to be gathered at this point. However, an unencrypted message is able to be read in its entirety, attributes and data.

The message next moves from the ServerInterceptor to the IncomingValidator class, which validates the security of the message. Checks are performed to ensure that the user is authorized, and that both signature and encryption were correctly applied to the message. If there is any check out of order or missing from the WS-Security header of the message it will be rejected and the transaction will end.

Once the message is successfully validated it moves to the ServerHandler class, which has access to the SOAP message and can access individual parts of the message directly. This is different from the raw XML access that is seen in the ServerInterceptor. Within the WASP environment the SOAP message is represented as a Document Object Model (DOM) object with methods that can access and manipulate the data within. Even if a message has been sent with encryption, it has been decrypted by the WASP server at this point, so we are able to gather data from the fields of the SOAP message (e.g. input parameters on request and output message on return).

After passing through the ServerHandler process, the message arrives at the actual web service where the method invocation is processed. For the return trip back to the client, the message passes back through the ServerHandler process and straight back to the client. It should be pointed out that these checkpoints are all contained within and belong to each service instance, they are not part of the general server environment.

Each web service instance has connections to three databases. The first is considered the primary source, this is the database that the service uses for storing its asset information. Access to

this database is read/write, enabling the service instance to access and update its own databases. The second two connections are considered alternate sources, corresponding to the other two service instances. Access to these databases is read only, controlled by user rights in the database server. From the perspective of a given user these alternate sources are used exclusively for calculating Data Consistency. Considering the ForwardWS instance, its primary connection is to its own database, and its alternate sources connect to the RearWS and HomeWS databases. These connections are illustrated in the previously listed Figure 4.3.

4.3 Trust Calculation

The aforementioned checkpoints are where trust is calculated as each message is processed. Each checkpoint calculates a subset of trustability indicators and stores them in a transaction log. The ServerInterceptor initiates a transaction entry in the log, and computes the Data Currency as well as Source Credibility factors. The IncomingValidator verifies and computes trust for the WS-Security indicators. The ServerHandler (incoming) captures the input parameters for the method being invoked; these are used for determining Data Consistency. On the outgoing side, the ServerHandler captures output generated by the method request and calculates trust for the Data Consistency Indicator. A summary of the checkpoints and their purpose is shown below in Table 4.2, as well as in the previous shown Figure 4.4.

Table 4.2: Role of Checkpoints in Trust Calculation

Checkpoint	Purpose
ServerInterceptor	Initiates transaction; calculates Data Currency, Source Credibility
IncomingValidator	Calculates WS-Security indicators
ServerHandler	Calculates Data Consistency

As mentioned in Section 4.1, additional tables are required in the database. Their function is to assist in the computation of information trustability. The first is a transaction log which records details about each method invocation (name of method, input parameters, and output returned), as well as the calculated trust for every transaction. The second stores the expected currency

values for each data type. The final table stores client preferences for indicator bracketing, as well as the source credibility rating. For illustrative purposes three scenarios are provided in the client interface so that a range of trust values can be assessed, but the values can be easily updated to reflect a clients true bracketing preferences. These additional tables are shown in Figure 4.5, with the full SQL DDL description listed in Appendix B.

expectedCurrency	
cr_id **(PRIMARY KEY)**	*VARCHAR(10)*
dataType	*VARCHAR(20)*
expCurr	*VARCHAR(2)*

clientProfile	
client_id **(PRIMARY KEY)**	*VARCHAR(10)*
w_wssSig	*VARCHAR(2)*
w_wssEnc	*VARCHAR(2)*
w_wssAuth	*VARCHAR(2)*
w_dataCons	*VARCHAR(2)*
w_srcCred	*VARCHAR(2)*
w_dataCurr	*VARCHAR(2)*
actSrcCred	*VARCHAR(2)*

txLog	
tx_id **(PRIMARY KEY)**	*VARCHAR(10)*
timestamp	*VARCHAR(15)*
srvBodyID	*VARCHAR(45)*
methodName	*VARCHAR(20)*
inputStr	*VARCHAR(255)*
outputStr	*TEXT*
t_wssSig	*VARCHAR(2)*
t_wssEnc	*VARCHAR(2)*
t_wssAuth	*VARCHAR(2)*
t_dataCons	*VARCHAR(2)*
t_srcCred	*VARCHAR(2)*
t_dataCurr	*VARCHAR(2)*

Figure 4.5: Additional Database Tables

4.3.1 Detailed Role of ServerInterceptor. As briefly mentioned earlier, the ServerInterceptor is what initiates the transaction entry. Each message has a unique identification number. This number is extracted from the raw XML data using the second group from the following regular expression: (<e:Body.*wsu:Id="Body-Id-)**([-a-fA-F0-9]*)**(">). An input string of <e:Body wsu:Id="Body-Id-0212d130-722e-11d9-b8d7-f300a1e0b8d7"> to this expression would yield an extracted ID of 0212d130-722e-11d9-b8d7-f300a1e0b8d7. This ID is stored in the transaction log so that the IncomingValidator and ServerHandler can update the log for the same message. A timestamp at message receipt is also taken, and compared to the timestamp of the last update. The difference between the two is combined with expected currency for the object type and stored

as the Data Currency value. Source Credibility is also calculated here by examining the specified credibility for the source and retrieving the clients preferred bracketing from the clientProfile table.

Initial values are assigned for the remaining indicators (WS-Security Authorization, WS-Security Signature, WS-Security Encryption, and Data Consistency). The values are the lowest possible for that indicator, `(00 + bracket)`. In the event that the transaction ends prematurely or something other than the desired outcome results, the trust values will remain at that low value. In the event that a method is not listed in the deployment descriptor file, it will be unsecure and there will be no body-id in the message. If this happens the body-id field in the transaction log is set to `"unsecure"`, and all of the initial low trust values will remain. Regardless of whether a secure or unsecure message generated low trust values, this should serve as an indicator to users of the system that trustability is legitimately degraded. For example, either the data has been incorrectly input, unsecured methods have been invoked, or a malicious user has bypassed the security mechanisms in place.

4.3.2 Detailed Role of IncomingValidator. The body-id number that was recorded by the ServerInterceptor is extracted from the message by the IncomingValidator so it can update the correct record in the transaction log. The message security configuration is parsed into individual components and checked to ensure that each one (authorization, signature, and encryption) was correctly applied. Each component is examined independently; if any component was correctly implemented, then the trust value is increased, otherwise it is decreased. As mentioned previously, a message that has not been properly secured will be rejected by the WASP server, meaning that it wouldn't make it to this point. In that case, the initial values set by the ServerInterceptor of the lowest trust possible would stand in the transaction log, interpreted later as a caution flag to users.

Even though the service has been configured to require maximum security, we still require and enforce these security checks. If high security is required and expected, then a sudden lapse in security can mean one of two things: a security hole has been unintentionally left in the system (e.g.

methods not specified in the deployment descriptor), or a security breach has been intentionally created by a malicious user. In either case, the trustability of the information within the system should be lowered. Again the values set initially by the ServerInterceptor provide that safety net.

4.3.3 Detailed Role of ServerHandler. The ServerHandler is composed of (among other things) a method that handles SOAP requests (`handleRequest()`), and a method that handles the response (`handleResponse()`). The state of the method invocation request is maintained between these two methods, such that the session is not complete until the response has been sent. This affords the opportunity for the body-id of the message to be extracted from the request, and held until the response, allowing the transaction log to be updated for the correct record. The single state also decreases the number of database transactions that must be made; data is gathered from the request method and held until all necessary data has been gathered from the response method. At that point the transaction log is updated once for that message recording the calculated Data Consistency value.

The main contribution of the ServerHandler is the calculation of the Data Consistency indicator. It is done at this point because the input to and output from the method can be seen here. Using the aforementioned alternate database connections, a query is built based on the type of object being accessed, the input supplied, and any output that was returned. Both of the alternate sources are queried for similar data, returning a SQL result set that is parsed to determine if there were any matches. As described in Section 3.4.3 and using Algorithm A.2, if a match is found, then the consistency bit for that source is set to `1`, otherwise it is set to `0`.

4.4 Trust Interpretation

As noted in Table 4.1, the `edu.afit.assetTracker.trust` package handles all of the trust interpretation. It contains an overall class (`edu.afit.assetTracker.trust.Trust`), and five extension classes for each type of indicator. There is no interface code in this package, it is purely an

interpretation engine that accepts trust values and outputs interpretations. The user interface is part of `edu.afit.AssetTracker.ui` as demonstrated in Section 4.6.

The main trust class accepts a hexadecimal trust value string as a constructor parameter. Upon instantiation of a trust object, the provided trust value is parsed into individual indicator values. Then a TreeMap is built holding appropriate trust objects (WS-Security, Source Credibility, Data Consistency, or Data Currency) with the indicator name as the key. The trust value must have indicators in the following order: WS-Security Authorization, WS-Security Signature, WS-Security Encryption, Data Consistency, Source Credibility, and Data Currency. This ordering is ensured by programatically building the trust value from the records in the transaction log table and by denying users the opportunity to manually enter trust values.

As each trust object is created, its interpretation value is automatically calculated and stored in the object. A getter method is available for retrieving and displaying the value as needed. The initial value is calculated with a normal tolerance value (`010`), but users can easily adjust the amount of desired tolerance in the provided interface. If tolerances are changed, then users can request a new calculation of the trust interpretation, and the new value is calculated as before. Because the Data Currency indicator does not have tolerance to account for due to the way it is represented, the Data Currency trust interpretation class does not have a mechanism for adjusting it, although an empty method is in place for future capabilities of re-calculating the interpretation.

4.5 Build Environment

The web services runtime environment used is Systinet WASP Server for Java. Since all developed code is compliant with Java 1.4.2_05, it ought to be able to be run on any platform for which there is a Java Virtual Machine and MySQL database available. Despite the promises of cross platform compatibility, it would come as no surprise if some unanticipated errors occur if

this code is executed on a different platform configuration. To establish a common baseline, the tools and versions listed in Table 4.3 should be used to guarantee the advertised performance.

Table 4.3: Development Environment Baseline

Product	Version	Purpose
Java™	1.4.2_05	Main development language
Systinet WASP Server for Java	5.0	Web services runtime environment
MySQL [5]	4.1.7	Relational database engine
Eclipse [3]	3.0.1	Java IDE used for all development except GUI building
Apache Ant [1]	1.6.1	Java build tool
NetBeans [6]	3.6	Java IDE used for GUI building
Mac OS X [2]	10.3.7	Operating system

In addition to the package hierarchy outlined in Table 4.1, there are several files used for compiling, deploying, and running the code. All files are listed from the project root. Since Ant is used as the build tool, `\build.xml` and `\build.properties` are used to invoke any actions. The `\dd` folder contains the previously mentioned deployment descriptors that tell the WASP server how to configure the service instance and set security parameters. The `\wsdl` folder is where WSDL files are stored when they are generated. If the entire project is opened and used within Eclipse, the build file can be executed using its integrated Ant tools. If the project is run from a command line interface, then the `\runant.sh` or `\runant.bat` (depending on the users operating sytem) files must be executed in place of the regular Ant command to setup necessary CLASSPATH and environment variables.

The Ant build file provided contains all necessary targets to compile, deploy, and run the service instances and clients "out of the box". All targets assume that the WASP server is already running at the address and port specified in the `\build.properties` file. The pertinent targets are listed and defined below in Table 4.4 where `atDBNAME` is equal to `atForward`, `atRear`, or `atHome`.

A series of SQL files are provided to create all necessary databases, tables, and initial data. The entire database environment can be reset to a clean initial condition by executing `master.sql`

Table 4.4: Key Ant Targets

Target	Purpose
create_identities	Creates the necessary identities and certificates for both the server and client. Must be run once before any of the example code will work properly.
make_*_service	Three targets exist, * indicates forward, rear, or home. Compiles the service, generates necessary WSDL files, and builds a JAR file for deployment to the server
deploy_*_service	Three targets exist, * indicates forward, rear, or home. Deploys the service to the specified service instance
undeploy_*_service	Three targets exist, * indicates forward, rear, or home. Undeploys the specified service instance
make_*_client	Three targets, * indicates forward, rear, or home. Compiles the client, retrieves the WSDL files from the deployed services, and builds a JAR file for execution
run_gui	Launches the interface for testing the code for this work

from a MySQL command prompt. The files and their respective functionality are listed below in Table 4.5.

Table 4.5: Database Initialization SQL Files

File	Purpose	Called By
master.sql	Calls all files, resets the database to a fresh initial state	N/A
createTables.sql	Creates the required tables for each database	atDBNAME/setupDBNAME.sql
databaseCreate.sql	Creates each database (atForward, atRear, and atHome)	master.sql
dataClientProfile.sql	Loads data for the client bracketing preferences	atDBNAME/setupDBNAME.sql
dataCurrency.sql	Loads default expected currency values for each data type	atDBNAME/setupDBNAME.sql
atDBNAME/setupForward.sql	Creates tables, loads expected currency, bracketing preferences, and sample data for each data type	master.sql
atDBNAME/dataAircraft.sql	Loads sample data for Aircraft objects	atDBNAME/setupDBNAME.sql
atDBNAME/dataLocation.sql	Loads sample data for MAJCOM and Bases objects	atDBNAME/setupDBNAME.sql
atDBNAME/dataMunition.sql	Loads sample data for Munition objects	atDBNAME/setupDBNAME.sql
atDBNAME/dataPersonnel.sql	Loads sample data for Personnel objects	atDBNAME/setupDBNAME.sql

4.6 Sample Output

As listed in Table 4.4, the `create_identities` must have been run before any code can be successfully executed. Assuming that it is has been run, and that the services are deployed to their

respective locations, we can begin executing the client code. The main AssetTracker interface is launched by executing the `run_gui` target and is shown in Figure 4.6.

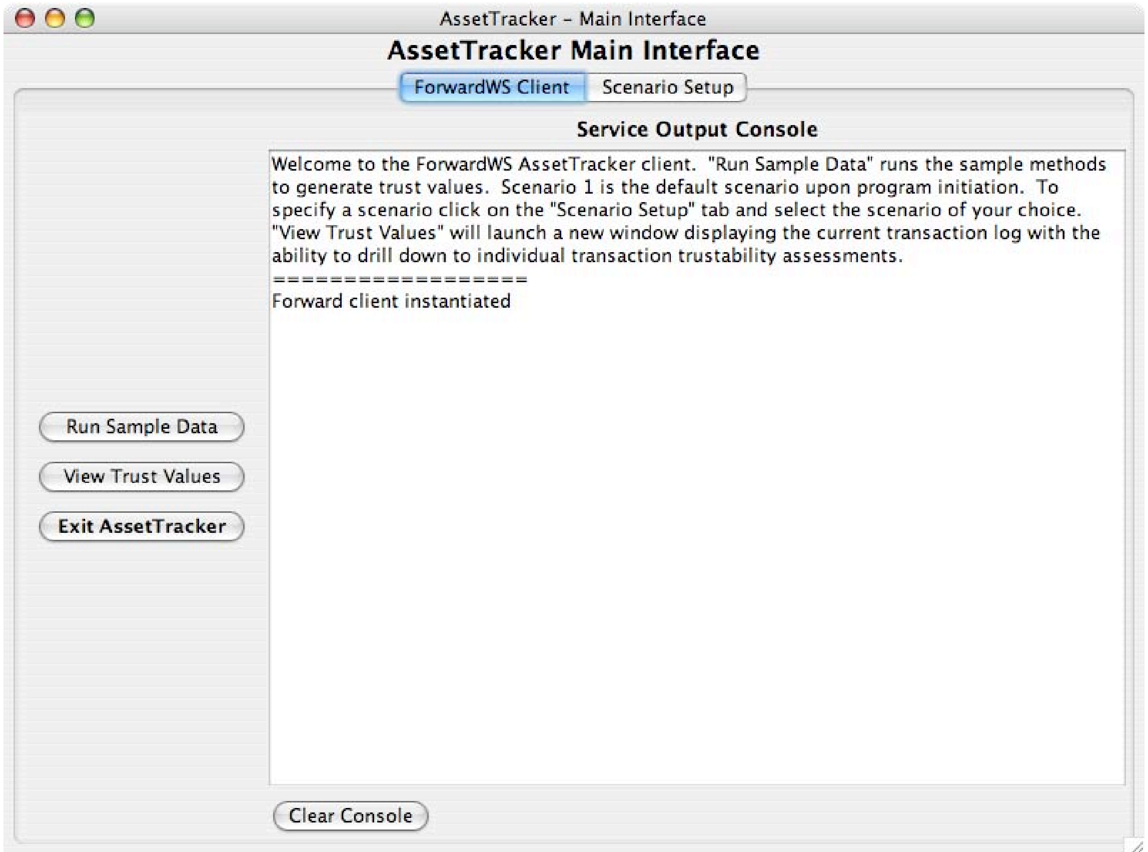

Figure 4.6: Initial Application Screen

In order to generate a range of trust values, three scenarios are available within the AssetTracker interface. Each scenario sets client bracket preferences and source credibility rating for the ForwardWS as shown in Table 4.6. As a reminder, bracket preferences range from narrow to wide, represented in binary from `00` to 11 and are listed in Table 3.1; source credibility ratings range from low to high, represented in binary from `00` to 11 and are listed in Table 3.3. Once a scenario has been selected, the user can execute a set of sample transactions. This invokes a sequence of methods for each object type in the following order: MAJCOM, Base, Aircraft, Munition, Personnel. Each object has a variety of get, delete, add, and update methods invoked in no particular order.

Table 4.6: Scenario Configurations

Scenario	w_wssAuth	w_wssSig	w_wssEnc	w_dataCons	w_srcCred	w_dataCurr	actSrcCred
1	11	11	11	01	10	00	01
2	11	11	11	00	10	11	11
3	10	10	10	11	01	10	00

Each scenario has an inherent influence on the trust values that are generated. Scenario Two(high) generates the highest values, given its total bracket preferences are higher than Scenarios One(medium) or Three(low), and its source credibility is the highest. Similarly Scenario Three(low) generates the lowest values since it has the narrowest bracket preferences. Scenario One(medium) is the default scenario, and generates trust values in between Scenarios Two(high) and Three(low).

Once the sample data transactions have been executed, the resulting transaction log can be viewed for analysis. This presents a table showing transaction ID, method name, input string, output string, and trust value from the transaction log as well as a calculated trust interpretation. This screen is shown in Figure 4.7.

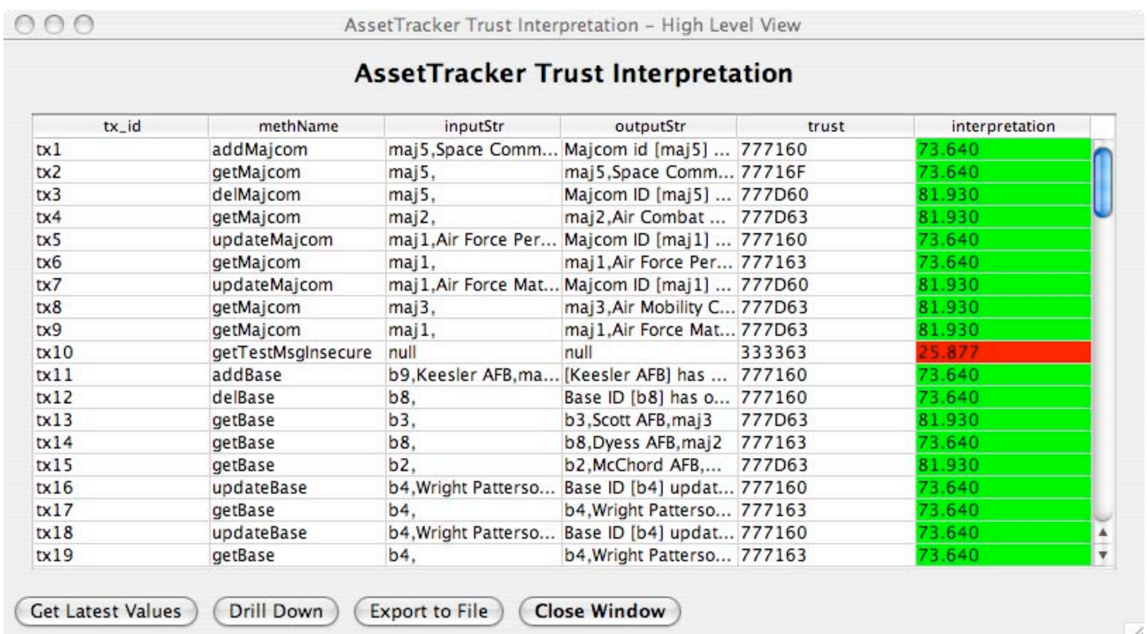

Figure 4.7: Trust Value Table

The interpretation values are calculated using a normal tolerance and color coded as described in Section 3.2. There is no room for tolerance adjustment on this screen, this is merely a high level view of the overall trustability assessment for each transaction. In order to adjust tolerance for a particular transaction, users can drill down for more information on the selected table row. Considering transaction `tx6` as shown in Figure 4.7, the resulting detailed view of the trust interpretation is shown in Figure 4.8.

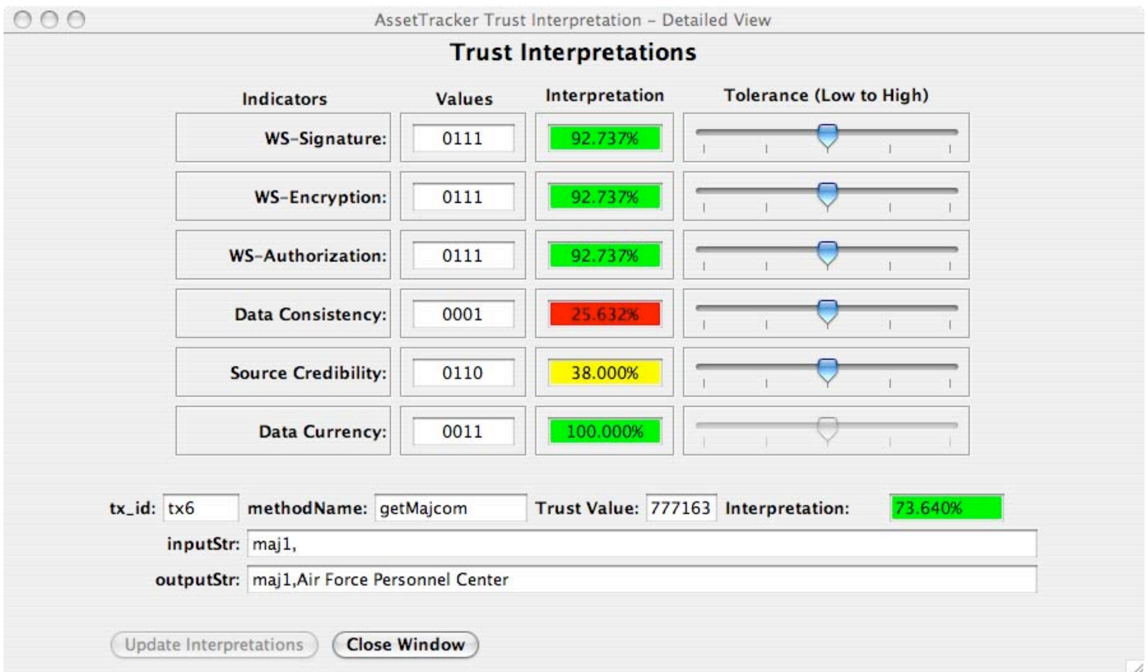

Figure 4.8: Detailed Trust Interpretation Display

In the detailed view, interpretation of individual indicators are color coded, as well as the interpretation for the entire transaction. These values and their color coding change as tolerance levels are adjusted and the interpretations are updated by the user. Continuing with transaction `tx6`, lowering all tolerance levels to the lowest setting results in the transaction interpretation changing from the initial interpretation of 73.640% to the lower 68.991% as shown in Figure 4.9. It should be noted that the color coding changed with the new interpretation since the value fell below the 70% boundary.

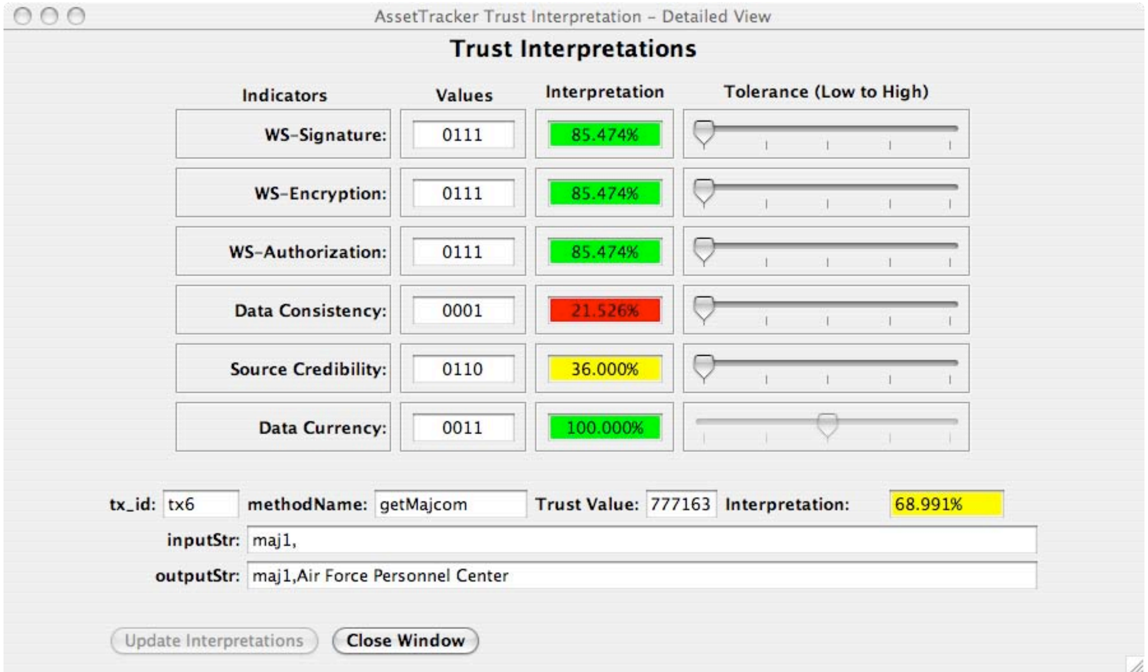

Figure 4.9: Detailed Trust Interpretation Display With Lowered Tolerance

4.7 Summary

This chapter has discussed the implementation that demonstrates the trustability assessment model proposed in Chapter III. AssetTracker meets all of the specifications, it repeatably calculates trust values, and provides interpretation values that can be adjusted for the amount of allowable tolerance. Three scenarios are utilized to generate a range of trust values. The results of the scenarios are discussed and analyzed in Chapter V.

V. Conclusions and Future Work

Chapter IV discusses and demonstrates the AssetTracker system as a representative example of how to create and utilize trust values in web services based data exchange. This chapter builds upon that and discusses the trust values generated by the three different scenarios and conjectures on their meaning. The success of the implementation to include its usefulness in a decision making environment is discussed. Finally, several areas for future enhancements to this work are presented.

5.1 Conclusions

Quality of information plays an increasingly important role in decision making environments. With this increased reliance comes the need to discriminate between trustable and *un*trustable information. The model proposed in this work provides this capability. In order to maintain a reasonable scope, focus has been limited to the secure communications domain of web services. When considering the overall effectiveness of the implementation, there are several areas that must be discussed. First, we examine whether or not the trust values are constructed in a repeatable way as described in Section 3.4. Second, the usefulness of the trustability assessment in a decision making environment is analyzed. Finally, the role that the three scenarios from Section 4.6 play in determining the overall utility of the system is discussed.

5.1.1 Repeatable Representation. As depicted in Figure 5.1 our trust values have been constructed in a repeatable manner. Each of the three scenarios has the same pattern, but different ranges of interpretation values. This is *not* a graph representative of the overall system trustability over time, rather it is a graph showing the individual trustability interpretations for a set of transactions. All interpretation values are calculated using a normal tolerance; adjustments made to individual indicator tolerance levels would change the actual plots for each scenario.

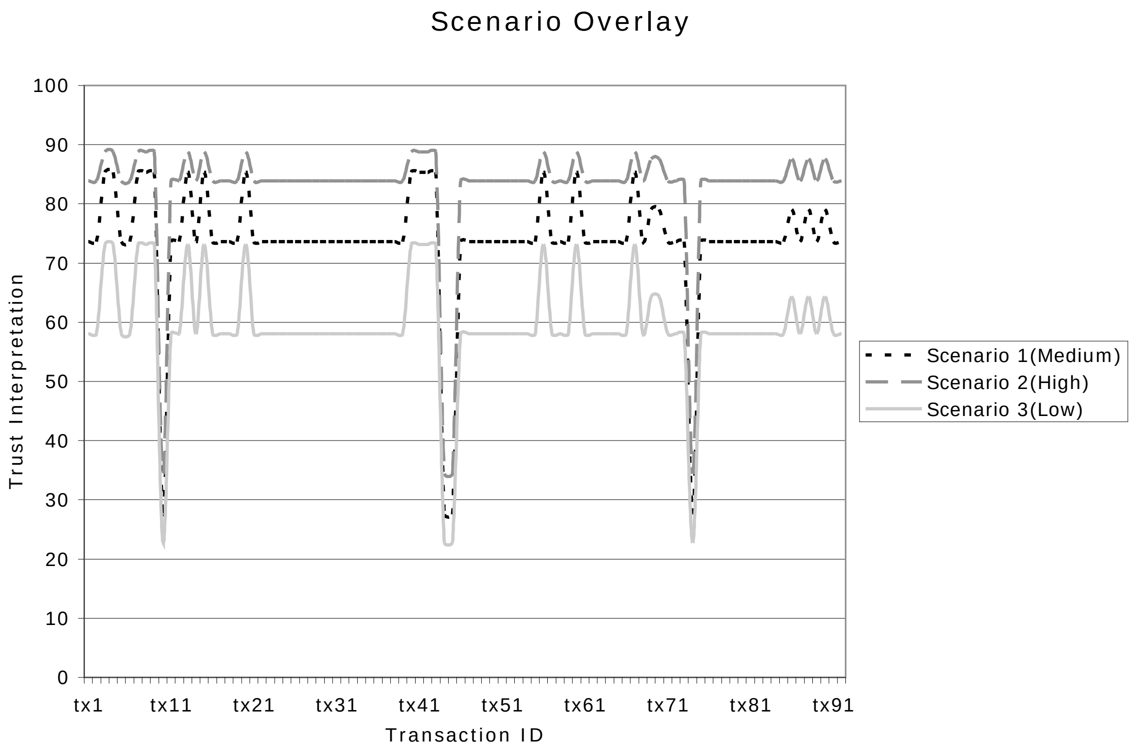

Figure 5.1: Scenario Trust Interpretations Overlaid

Each scenario makes the same method calls (retrieval, addition, update, deletion), on the same initial state of the database. Repeated execution on the same initial state generates the same set of trust values every time. It follows that the trust values are the same for each scenario aside from the differing brackets and source credibility. The bracket and source credibility variations are what account for each scenario raising or lowering the trust values according to the configuration as described in Section 4.6.

5.1.2 Usefulness in Decision-Making Environment. The goal of this work is not to prevent untrustable information from entering the system, but rather to alert decision makers to the presence of any. Knowledge of the presence of untrustable information allows them to react appropriately to the decision at hand. Unsecured methods are accessed several times during each scenario. As seen from the low points in Figure 5.1, the trustability is severely degraded when

they are accessed, alerting users to the fact that something has legitimately lowered the trust. A detailed examination of the transaction log will reveal that an unsecure method was invoked. Further investigation can be made to determine if the unsecured method was intentionally allowed, if it was mistakenly allowed, or if a malicious user was able to gain unauthorized access to it.

Although the impact of trust assessment values is conveyed visually here, there is a clear foundation for a machine-based decision making system to process and act upon the values. Due to the pre-defined format of trust values, each value can be parsed to determine what set of inputs resulted in its value. Using transaction `tx7` from Figure 4.7, we can parse its trust value of `777D60`. Table 5.1 shows how the value is parsed.

Table 5.1: Trust Value Parsing - `777D60`

Indicator Value	Binary Value	Meaning
7	0111	WS-Security Authorization was successfully applied; wide bracket
7	0111	WS-Security Signature was successfully applied; wide bracket
7	0111	WS-Security Encryption was successfully applied; wide bracket
D	1101	Data was consistent with both alternate sources; medium-narrow bracket
6	0110	Source credibility is medium-low; medium-wide bracket
0	0000	Actual currency is very current; expected currency is very current

All trust assessment values are parsed the same way. There is a finite set of possible ways that any given trust value can have been constructed. From this we can see how it is easy to construct an algorithm that can react appropriately to any trust value. There is nothing that prohibits a dual approach, where an automated system uses the trust assessments to aid its decision making, and a human user interface exists alongside allowing views into the trustability of information.

5.1.3 Impact of Bracket Selection and Source Credibility. Scenario Three (see Table 4.6) has very narrow brackets for the WS-Security indicators, as well as a very low source credibility rating. The overall trust value interpretations are lower than the other scenarios as expected. The narrowing of brackets has the effect of decreasing the overall trustability for the service instance. Scenario Two reverses that trend by using very wide brackets, and a high source credibility rating.

Figure 5.2 shows trust interpretations from each scenario in consecutive order (Scenario One, Scenario Two, Scenario Three). Since the methods executed are the same, and occur in the same order for each scenario, we can see that the transaction trust assessments are increased or decreased in proportion to the value of the brackets and source credibility. As expected, Scenario One (having medium brackets and source credibility values) generated interpretations greater than Scenario Three, but less than Scenario Two.

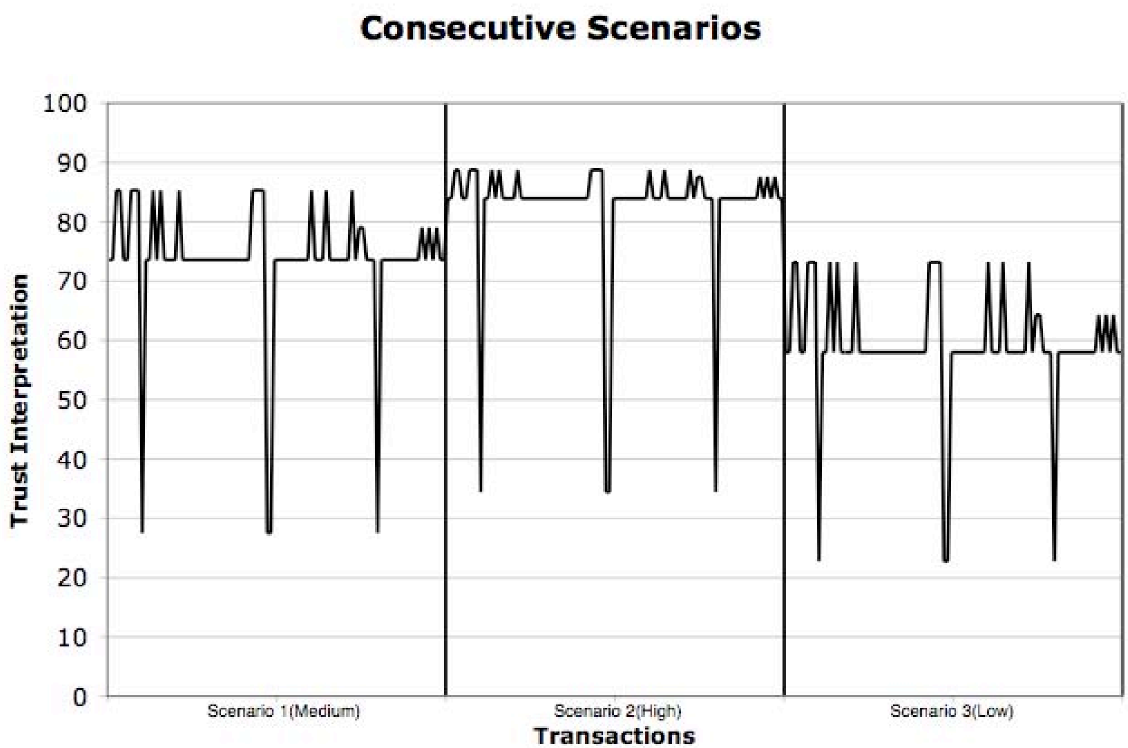

Figure 5.2: Scenario Trust Interpretations Consecutively

5.2 Future Work

As shown in Chapter II, there is significant interest throughout the literature and industry in assessing trustability of information. As discussed in the previous section, the model and methodology proposed in this work attempts to provide improvements over existing trust assessment models. Despite the progress of the present work, there are additional features that could further

enhance the value provided in a decision making environment. These potential enhancements are outlined and reviewed here.

5.2.1 Multiple Clients. As detailed in Chapter IV, the trust values generated in this work use a single client instance with differing bracket and source credibility scenarios. While this provides a broad range of trust values, an enhanced implementation could use multiple client instances. In addition to providing the same insight as the approach of this work, multiple clients allow for the notion of seeing how other service instances rate each others trustability. Of particular interest when assessing Data Consistency and Source Credibility, it would be interesting to see how each instance rates the consistency and credibility of other sources, in effect building up a default trust bias. For example, if *ForwardWS* generates a Data Consistency value indicating the data is consistent among all sources for a transaction involving aircraft `a3`, will *RearWS* and *HomeWS* generate the same value? Implications of these differences on the decision maker could be explored.

5.2.2 Database Replication. The initial data used for this implementation is similar across all service instances with the exception of some minor differences introduced to obtain a range of trustability assessments. Data is not explicitly replicated from one database to another. If multiple clients are used, unless they are configured to initiate similar transactions (i.e. exact replications) for each service instance, the databases will only increase in their differences. This guarantees that Data Consistency values will decrease as transactions continue to occur. Data replication would solve this problem, increasing the usefulness of the trust assessments.

5.2.3 Distributed System Integration. The AssetTracker system was built specifically to demonstrate the model proposed in Chapter III. While it serves that purpose well, the true test of the present works' trust assessment methodology is how well it would function in a real world system. Much in the same way that AssetTracker is able to calculate trustability assessments, any

SOAP web service environment should be able to accommodate a set of trustability extensions. Real world data and a sizable transaction load would provide insight into the effectiveness of this work. The Joint Battle Infosphere from the Air Force Research Laboratory would be a good platform to investigate for this integration as they are in the process of moving to a web services based platform.

5.2.4 Transaction Log Data Mining. We have limited our use of the transaction log to merely gather trust assessment values and calculate interpretations. Despite this self imposed restriction, there is a wealth of information that can be gleaned. Logs can be searched for unusual patterns in method access, e.g. an extraordinarily large amount of invocations for a particular object. These (and other) anomalies could be correlated to the resulting trust values. The effect that certain method invocations have on trust values could be determined from the logs, perhaps giving additional insight into the overall information trustability.

5.3 Summary

This work addresses the issues that exist in assessing information trustability. The proposed solution is framed in a secure web services environment, and uses a series of indicators and attributes to derive trust values for information from web service transactions. The values are constructed in a repeatable manner such that the value can be parsed by humans or machines to determine its meaning. Assessment values are translated to a percentage-based interpretation that enhances the the ability of the decision maker to determine the basis for confidence in presented information.

Appendix A. Key Algorithms

This appendix contains the full algorithms in pseudo-code for calculating Data Currency and Data Consistency indicators. Algorithm A.1 contains the Data Currency algorithm, and is discussed in Section 3.4.5. Algorithm A.2 contains the Data Consistency algorithm and is discussed in Section 3.4.3.

Algorithm A.1 Determining Data Currency

```
if actualCurrency == "00" then
  interpretation = 100
else if actualCurrency == "01" then
  if expectedCurrency == "00" then
    interpretation = (100/3) * 2 = 66.66
  else
    interpretation = 100
  end if
else if actualCurrency == "10" then
  if expectedCurrency == "00" then
    interpretation = (100/3) = 33.33
  else if expectedCurrency == "01" then
    interpretation = (100/3) * 2 = 66.66
  else
    interpretation = 100
  end if
else if actualCurrency == "11" then
  if expectedCurrency == "00" then
    interpretation = 0
  else if expectedCurrency == "01" then
    interpretation = (100/3) = 33.33
  else if expectedCurrency == "10" then
    interpretation = (100/3) * 2 = 66.66
  else
    interpretation = 100
  end if
end if
```

Algorithm A.2 Determining Data Consistency

```
if message reaches Handler then
  get methodName being invoked /* from transaction log */
  get method input /* from transaction log */
  if methodName starts with "get" then
    for all datatypes do
      SELECT * FROM alternateSource1 WHERE id = inputID
      for all query results do
        if all attributes match then
          Consistency += "1"
        else
          Consistency += "0"
        end if
        SELECT * FROM alternateSource2 WHERE id = inputID
        if all attributes match then
          Consistency += "1"
        else
          Consistency += "0"
        end if
      end for
    end for
  else if methodName starts with "add" or "update" then
    for all datatypes do
      SELECT * FROM alternateSource1 WHERE allAttributes = allInputAttributes
      if all attributes match then
        Consistency += "1"
      else
        Consistency += "0"
      end if
      SELECT * FROM alternateSource2 WHERE allAttributes = allInputAttributes
      if all attributes match then
        Consistency += "1"
      else
        Consistency += "0"
      end if
    end for
  else if methodName starts with "delete" then
    for all datatypes do
      SELECT * FROM alternateSource1 WHERE id = inputID
      if there are no results then /* deleted data does not exist */
        Consistency += "1"
      else
        Consistency += "0"
      end if
      SELECT * FROM alternateSource2 WHERE id = inputID
      if there are no results then /* deleted data does not exist */
        Consistency += "1"
      else
        Consistency += "0"
      end if
    end for
  end if
end if
```

Appendix B. SQL DDL Descriptions

This contains the SQL DDL descriptions for each AssetTracker database table.

Table B.1: MAJCOM SQL DDL Description

Fieldname	SQL Type	Description
`majcom_id` (primary key)	`VARCHAR(10)`	ID for the MAJCOM
`majcom_name`	`VARCHAR(40)`	Name of the MAJCOM

Table B.2: Base SQL DDL Description

Fieldname	SQL Type	Description
`base_id` (primary key)	`VARCHAR(10)`	ID for the base
`majcom_id` (foreign key)	`VARCHAR(10)`	ID of the parent MAJCOM
`base_name`	`VARCHAR(25)`	Name of the base

Table B.3: Aircraft SQL DDL Description

Fieldname	SQL Type	Description
`aircraft_id` (primary key)	`VARCHAR(10)`	ID of the aircraft
`base_id` (foreign key)	`VARCHAR(10)`	ID of the parent base
`model`	`VARCHAR(20)`	Model of the aircraft (e.g. F-16)
`status`	`VARCHAR(20)`	Status of the aircraft

Table B.4: Munition SQL DDL Description

Fieldname	SQL Type	Description
`munition_id` (primary key)	`VARCHAR(10)`	ID of the munition
`base_id` (foreign key)	`VARCHAR(10)`	ID of the parent base
`name`	`VARCHAR(20)`	Name of the type of munition
`status`	`VARCHAR(20)`	Status of the munition

Table B.5: Personnel SQL DDL Description

Fieldname	SQL Type	Description
`personnel_id` (primary key)	`VARCHAR(10)`	ID of the personnel
`base_id` (foreign key)	`VARCHAR(10)`	ID of the parent base
`title`	`VARCHAR(20)`	Title of the personnel
`status`	`VARCHAR(20)`	Status of the personnel

Table B.6: clientProfile SQL DDL Description

Fieldname	SQL Type	Description
`client_id` (primary key)	`VARCHAR(10)`	ID tag for the client (e.g. Forward)
`w_wssSig`	`VARCHAR(2)`	Bracket for WS-Security Signature
`w_wssEnc`	`VARCHAR(2)`	Bracket for WS-Security Encryption
`w_wssAuth`	`VARCHAR(2)`	Bracket for WS-Security Authorization
`w_dataCons`	`VARCHAR(2)`	Bracket for Data Consistency
`w_srcCred`	`VARCHAR(2)`	Bracket for Source Credibility
`w_dataCurr`	`VARCHAR(2)`	Bracket for Data Currency
`w_actSrcCred`	`VARCHAR(2)`	Source Credibility rating for the source

Table B.7: expectedCurrency SQL DDL Description

Fieldname	SQL Type	Description
`cr_id` (primary key)	`VARCHAR(10)`	Currency table ID
`dataType`	`VARCHAR(20)`	Data type for currency value
`expCurr`	`VARCHAR(2)`	Expected currency for data type

Table B.8: txLog SQL DDL Description

Fieldname	SQL Type	Description
`tx_id` (primary key)	`VARCHAR(10)`	Transaction ID
`timestamp`	`VARCHAR(15)`	Timestamp of the message
`srvBodyID`	`VARCHAR(45)`	Body ID of the message being logged
`methodName`	`VARCHAR(20)`	Name of method invoked
`inputStr`	`VARCHAR(255)`	Comma delimited list of method input parameters
`outputStr`	`TEXT`	Output returned by the invoked method
`t_wssSig`	`VARCHAR(2)`	Trust value for WS-Security Signature
`t_wssEnc`	`VARCHAR(2)`	Trust value for WS-Security Encryption
`t_wssAuth`	`VARCHAR(2)`	Trust value for WS-Security Authorization
`t_dataCons`	`VARCHAR(2)`	Trust value for Data Consistency
`t_srcCred`	`VARCHAR(2)`	Trust value for Source Credibility
`t_dataCurr`	`VARCHAR(2)`	Trust value for Data Currency

Bibliography

1. "The Apache Ant Project". URL `http://ant.apache.org`.
2. "Apple Mac OS X". URL `http://www.apple.com/macosx`.
3. "Eclipse Foundation". URL `http://www.eclipse.org`.
4. "Java Programming Language". URL `http://www.java.com`.
5. "MySQL Relational Database". URL `http://www.mysql.com`.
6. "NetBeans IDE". URL `http://www.netbeans.org`.
7. "Systinet WASP Server for Java". URL `http://www.systinet.com/products/ssj/overview`.
8. *SOAP Version 1.2 Part 1: Messaging Framework*. Technical report, W3C, June 2003.
9. *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*, 2004. URL `www.oasis-open.org`.
10. "Google News", 2005. URL `http://news.google.com/news?q=soap+web+services`.
11. "Yahoo News", 2005. URL `http://search.news.yahoo.com/search/news?c=&p=soap+web+services`.
12. Albrecht, Conan C. "How Clean is the Future of SOAP?" *Communications of the ACM*, 47(2):66–68, February 2004.
13. Bariff, Martin and Salimol Thomas. "An Operational Framework for Information Integrity". Feb 2003. URL `www.informationintegrity.org`.
14. Bhatti, Rafae, Elisa Bertino, and Arif Ghafoor. "A Trust-based Context-Aware Access Control Model for Web-Services". *Web Services, 2004. Proceedings. IEEE International Conference on*, 184–191. 6-9 July 2004 2004.
15. Chen, Peter P. *Information Validity Assessment and Meta Data Modeling in Integrating Heterogeneous Data Sources*. Technical report, Louisiana State University, 2001.
16. Clune, Jim and Adam Kolawa. "Security Issues with SOAP". *CrossTalk: The Journal of Defense Software Engineering*, July 2002. URL `http://www.stsc.hill.af.mil/crosstalk/2002/07/clune.html`.
17. Geer, David. "Taking Steps to Secure Web Services". *Computer*, 36(10):14–16, October 2003.
18. Hada, Satoshi and Hiroshi Maruyama. *SOAP Security Extensions*. Technical report, Tokyo Research Laboratory, IBM Research, 2000. URL `http://www.trl.ibm.com/projects/xml/soap/wp/wp.html`.
19. IBM and Microsoft. *Security in a Web Services World: A Proposed Architcture and Roadmap*. Technical report, IBM Corporation and Microsoft Corporation, 2002.
20. Kearney, P, J Chapman, N Edwards, M Gifford, and L He. "An Overview of Web Services Security". *BT Technology Journal*, 22(1):24–42, Jan 2004.
21. Khare, Rohit and Adam Rifkin. "Weaving a Web of Trust". *World Wide Web Journal*, 2(3):77–112, 1997.
22. Nayar, Madhavan K. "Information Integrity: the Next Quality Frontier". *Total Quality Management & Business Excellence*, 15(5,6):743–751, Jul/Aug 2004.

23. Pipino, Leo L, Yang W Lee, and Richard Y Wang. "Data Quality Assessment". *Communications of the ACM*, 45(4ve):211–218, April 2002.

24. Schneier, Bruce. "Crypto-Gram Newsletter - SOAP". *Crypto-gram*, June 2000. URL `http://www.schneier.com/crypto-gram-0006.html#SOAP`.

25. Staab, Steffen, W van der Aalst, V.R. Benjamins, A Sheth, J.A. Miller, C Bussler, A Maedche, D Fensel, and D Gannon. "Web Services: Been There, Done That?" *IEEE Intelligent Systems*, 18(1):72–85, January/February 2003.

26. Wagner, Ray. "Web Services Security Advances with Approval of Key Standard", 13 April 2004 2004. URL `http://www3.gartner.com/DisplayDocument?ref=g_search&id=430318`.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
03-21-2005	**Master's Thesis**	25 Aug 2004 – 21 Mar 2005

4. TITLE AND SUBTITLE

Assessing Information Trustability in a Secure Web Services Environment

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Penner, Charles, G., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/05-14

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFSE
Attn: Mr. Richard Metzger
Bldg 3, 525 Brooks Rd
Rome, NY 13441-4505 DSN: 587-7652

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Decisions are made based on available information. A decision support system endeavors to provide information that is timely, accurate, and trustable. Information gathered from secure web service transactions has attributes that can be used to assess a level of trustability. The trust assessments enable a decision maker to determine a basis for confidence in the information presented from the web service. Existing trust assessment models do not provide a way to determine from a particular trust assessment what information attributes contributed to its computation. The present work creates trust values that retain and denote meaning, allowing a decision maker to see specifically what factors influenced the information trust assessment. Also central to this work is interpretation of the trust assessments. The interpretation model allows users to specify the amount of allowable tolerance for reduced trustability in the decision being made. This "dial-a-trust" allows the interpretation to be scaled relative to the impact of the decision. The trust assessment values, along with their interpretations, allow both human and machine-based decision makers to determine whether information is trustable enough for the needs of the decision being made.

15. SUBJECT TERMS

Decision aids; decision support systems; information processing; information retrieval; information trustability; trust assessment; trust interpretation.

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
REPORT	**ABSTRACT**	**c. THIS PAGE**			Michael L. Talbert, Lt Col, USAF (ENG)
U	U	U	UU	82	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4613; e-mail: michael.talbert@afit.edu